1989

A comparative study of the performance of various image analysis methods for dimensional inspection with vision systems

Ralf Koeppe
Portland State University

AN ABSTRACT OF THE THESIS OF Ralf Koeppe for the Master of Science in Mechanical Engineering presented July 18, 1989.

Title: A Comparative Study of the Performance of Various Image Analysis Methods for Dimensional Inspection with Vision Systems

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:

[redacted]

Faryar Etesami, Chair

[redacted]

Hormozd Zarefar

[redacted]

Roy Koch

Dimensional inspection with Vision Systems requires a careful selection of image analysis methods in order to obtain accurate information about the geometry of the parts to be measured.

The purpose of this project is to study, implement and compare different image evaluation methods and to show their strengths and

weaknesses with respect to dimensional inspection. Emphasis is made on the inspection of circular features. The criteria of comparison for these methods are discussed. Using synthetically generated images, various analysis methods are compared and conclusions for their use are drawn. Results of the comparison show that the selection of a method has to be done with regard to the noise level of the measurement. Finally, a computationally fast calibration algorithm is studied and implemented.

A COMPARATIVE STUDY OF THE PERFORMANCE OF VARIOUS IMAGE ANALYSIS METHODS FOR DIMENSIONAL INSPECTION WITH VISION SYSTEMS

by

RALF HEINRICH KOEPPE

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
MECHANICAL ENGINEERING

Portland State University
1989

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of Ralf H. Koeppe presented July 18, 1989.

[redacted]

Faryar Etesami, Chair

[redacted]

Hormozd Zarefar

[redacted]

Roy Koch

APPROVED:

[redacted]

Graig A. Spolek, Head, Department of Mechanical Engineering

[redacted]

C. William Savery, Interim Vice Provost for Graduate Studies and Research

ACKNOWLEDGEMENTS

I wish to extend my warmest appreciations:

To Professor Faryar Etesami for his encouragement and guidance, and for giving me the opportunity to be creative;

To all faculty and staff members of the Department of Mechanical Engineering for their support;

To Dr. Rueff of the Institute of Manufacturing and Automatisation at the University of Stuttgart for his help and suggestions;

To my parents, Helga and Alfred, and my grandmother, Wilhelmine, for their love and support and for instilling in me the importance of education and helping me to achieve it;

And finally to Achim, André, Axel, Christina, Friedel, Raj, and Ray, and all my other friends in Portland, for their friendship.

Portland, Oregon

Ralf H. Koeppe

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

One research focus at Portland State University is Geometric Modeling and Tolerancing and Computer Aided Design. Recent research has been done on the theory of assembly verification of features by simulation or soft-gaging [1]. For the assembly verification process, design specification data has to be matched with feature data of the manufactured parts. The design specifications are obtained by using a solid model or a 2-dimensional drafting model of a design in a CAD - Systems. "Real World Data" of the manufactured part can be measured with the aid of coordinate measuring machines or vision systems. Where the former involves high cost and slow processing time but highly accurate results, the latter method allows faster processing time and lower cost.

Automated inspection with vision systems is being used in several industrial fields for the purpose of quality control. For example, the electronics industry uses vision systems to inspect miniaturized electronic components, which are beyond human inspection capability. Other systems are used to inspect automobile parts, metal surfaces, or food and packaging goods [2].

With the development of high resolution cameras and more powerful computers, vision systems are now used for dimensional inspection. The camera of the vision system takes an image of the part, which is then evaluated by image analysis and image processing methods. Geometric

parameters, such as diameter, length, width, and tolerances are calculated and then compared to the specification of the parts. The requirements for vision systems for dimensional inspection are high speed, flexibility, and high accuracy [3]. The computation time to process the image and to extract the required data must be high enough to match the speed of the production line. The system has to be flexible enough to be adjusted to a wide range of measurement conditions, such as lighting or individual characteristics of the part. Finally, vision systems have to be accurate enough to accomplish the task of dimensional inspection. Some systems use high magnification lenses, combined with accurate positioning tables, to obtain the required resolution for measurements with high accuracy [4][5]. However, for a lot of parts with large dimensions, the use of positioning tables is not appropriate. The entire feature of the part to be measured has to appear in the image, and therefore the resolution of these systems is lower than of the ones described above. This leads to the following questions: 1) How accurate can vision systems be without using positioning tables? 2) What requirements must such a system meet? and 3) Which methods of analyzing images must be applied to obtain satisfying results? Figure 1 shows such a system.

A part on a conveyor belt is transported below the camera. An image is acquired and the information about dimensions and tolerances of the part are extracted and processed by the frame grabber and edge detector. The features of the part are sampled and compared to the specification.

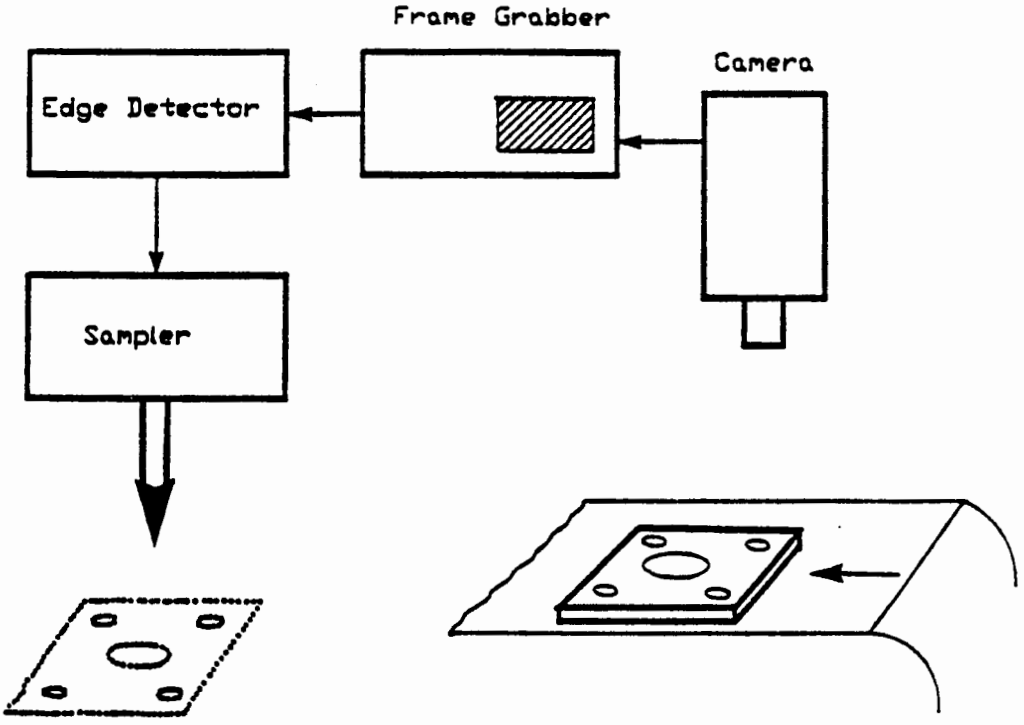


Figure 1. Automatic dimensional inspection system for tolerance verification.

1.1 BACKGROUND

Fundamental research has been done in image analysis and pattern recognition in the field of artificial intelligence. During the past 20 years, the use of vision systems has increased drastically and research in theoretical methods to extract information from the image has been strongly emphasized. Many comprehensive books have been published, and the newest achievements of research are discussed primarily in the journals, "Computer Vision, Graphics and Image Processing," "Pattern Recognition," "IEEE Transactions on Pattern Analysis and Machine Intelligence."

One main subject in this field is image processing. From a given scene, the two dimensional image representation is analyzed. One type

of image analysis method extracts features such as edges, others label, organize, and manipulate this detected information. The former methods are called image enhancement and edge detection methods. Image enhancement is the field of manipulating the contrast of an image, and smoothing and sharpening it. This can be done within the spatial domain or the frequency domain. The purpose of the edge detection methods is to extract features of an image.

The original images consist of a two dimensional area represented in image points called pixels. Using modern equipment, every pixel can assume 256 different gray values. This gray scale image can be converted into binary space, which results in a black and white image. Due to the complexity of the subject, a lot of research is still being done in developing methods for image analysis. References [3][6] - [8] provide insight into the use of image analysis methods, and are highly recommended for reading.

Related to the project are the publications by Peli and Malah [9], and Vliet, Young and Backers [10]. In both works, edge detection algorithms and the measures of comparing them are studied on circular features. However, this research and others focus on the general behavior of these algorithms. To determine how accurate and applicable image processing methods are for dimensional inspection with vision systems, the algorithms have to be implemented and tested for this specific task.

This project investigates the use of some fundamental image analysis methods, with regard to the measurement of circular features and their parametrization into geometric space. Among other works in

the field of image processing methods applied the application dimensional inspection, the publication by Rueff and Melchior [11] in the building of a software library is recommended for study. Another interesting work has been done by Petkovic, Niblack and Flickner [12] in developing high accuracy methods for the measurement of straight line edges.

Because of the enthusiasm and productivity of the research community in the field of artificial intelligence, it is necessary for engineers concerned with the application of image analysis methods to be up-to-date. New or improved algorithms have to be implemented, applied and compared for the special task of a specific application.

1.2 OBJECTIVES

As previously mentioned, the purpose of this project is an investigation of the application of image analysis methods for dimensional inspection with vision systems. Emphasis is made on the analysis of circular features. Operations in the spatial domain are investigated. Among the selected methods are: thresholding, in other words, the conversion from gray scale into binary images using different mapping operations; pixel gradient and variance methods, measuring the change of the gray values within an image; and methods of improving the image quality by preprocessing it. Image enhancement operations in the frequency domain are not considered in this work. The reason for this is that these operations can be similarly performed in spatial domain.

The image noise due to low image quality and variability of the

vision system is studied. Measures for comparing methods are defined. Using synthetical images, the methods are compared and the results are analyzed. These results are supported by applying and comparing these methods on real world images. Recommendations towards the use of image analysis methods are made.

To use vision systems for measurements, the system needs to be calibrated. A calibration model is studied and implemented. This calibration technique considers camera position and orientation, and lens distortion effect.

Finally, a model is proposed which shows that the image analysis method, the calibration model, and the resolution of a vision system determine its accuracy.

CHAPTER II

DIGITAL IMAGE PROCESSING

2.1 COMPONENTS OF A DIGITAL IMAGE PROCESSING SYSTEM

Image signals are generated at a video source, for example, a Charge Couple Devices (CCD) camera. Charge Couple Devices consist of light sensitive elements lined up in rows and columns. As soon as light illuminates an element, an analog signal is generated. The analog signal is then transformed to a digital format. In the digitalization process, samples of the analog signal at discrete time intervals are taken. The image is stored in the frame memory as a 2 - dimensional array, with one array element called a pixel. Each pixel is stored by an 8-bit value, which corresponds to 256 possible gray values. The frame memory is connected to the CPU of a microcomputer. By using software implemented on the microcomputer, several operations can be performed on the image stored in the frame memory. The image is displayed on a monitor by backtransforming the digital information using a display logic. Figure 2 shows how different components of an image processing system are connected.

Experiment Equipment for Digital Image Processing

The image processing equipment used in this research project consists of a PC-AT with a PCVISIONplus Frame Grabber [13], a digitizer, a CCD camera Pulnix TS - 545 and a display monitor. The

frame memory size is 1024 x 512 pixels with 8 bits per pixel. Either two images with 512 x 512 pixels or one single image with 640 x 480 pixels can be stored. The experiments were performed using the single image option. The input and output channels contain eight Look-Up Tables (LUT) each. They can be programmed to perform point operations on pixels in real time. In addition, PCVISIONplus provides software to interact with the frame grabber. These subroutines are written in C programming language and allow initialization, read-write operations, and the manipulation of the image. Software developed during this project is written in C, using Microsoft C 5.1.

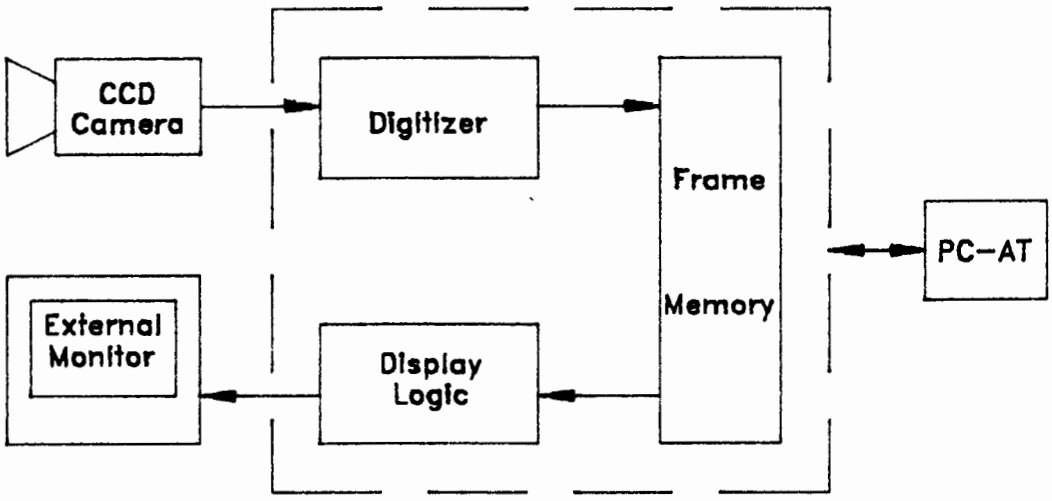


Figure 2.Components of a digital image processing system.

2.2 NOISE

In the field of vision system and image processing, the term "noise" describes the phenomena of the information content of an image, which obstruct the extraction of the desired information. For example, due to sampling and quantization errors and the sensitivity of the Charge Couple Devices (CCD), the intensity of the pixels vary for the same image scene. This variation is called system noise. The distribution of the system noise can be approximated by a Gaussian curve [3].

Other types of noise and their origin are shown in Table I.

TABLE I

TYPES OF NOISE

Noise	Origin	Type
System Noise	Hardware	additive random normally distributed with mean 0
Texture edge	Image	can be simulated by random normally distributed or by binary noise
Shadow	Lighting Conditions	not random noise

Noise introduces variation in the image. Therefore images with high system noise and texture edges require special image analysis methods. The occurrence of shadows can be avoided by providing symmetrical illumination of the scene.

A measure of how much image information is distracted by noise is the Signal to Noise Ratio (SNR) [9], defined as

$$SNR = \frac{h^2}{\sigma_n^2}$$

where h is the heighth of the edge, and σ_n the standard deviation of the noise. In the case of dimensional inspection the heighth of the edge can be approximated by

$$h \approx | \bar{I}_b - \bar{I}_o |$$

where $\bar{I}_b$ is the average intensity of the background, and

$\bar{I}_o$ is the average intensity of the object.

For high contrast images the Signal to Noise ratio is SNR --> ∞. For low contrast images SNR is a small value.

Intensity Variation of Pixels

To determine the variation in the intensity of an 8 bit pixel due to random system noise, and to investigate how image analysis techniques are effected by it, the following experiments were performed.

100 sample images of a cylindric black part on white background (SNR=2500) were studied and the intensity values of one object, one background, and one edge pixel were captured. The standard deviation and the range of the intensity values were calculated. In addition, the intensity variation of pixels processed by Roberts edge filter, pixel

averaging, and the mean and the variance of a n x n neighborhood of the processed pixels, were recorded. (For description of these methods see the end of this chapter.)

The experiment results in Table II demonstrate that the image noise is approximately normally distributed with a mean of zero (Figure 3). The standard deviation of the intensity of edge pixels is higher than the standard deviation of the intensity of object or background pixels. By using image averaging, the intensity variation decreases. This agrees with statistic theory. The process of averaging reduces the standard deviation of the intensity of a pixel by $(\sqrt{n})^{-1}$. Therefore, this technique can be used as a noise filter for random system noise. Noise is also decreased by calculating the mean of a pixel within its n x n neighborhood (Figure 4). Whereas image averaging requires the use of multiple images of a stationary object, the calculation of the local mean can be done on one image. The intensity variation of pixels processed by the variance or the Roberts edge filter is shown in Table II.

TABLE II

PIXEL INTENSITY VARIATION FOR DIFFERENT TECHNIQUES
STANDARD DEVIATION AND (RANGE) OF 100 SAMPLES

TECHNIQUE	OBJECT	BACKGROUND	EDGE PIXEL
GSI	2.03 (10)	1.64 (9)	2.29 (13)
GSI AVG 2	1.34 (5)	1.04 (5)	2.05 (10)
GSI AVG 4	0.92 (4)	0.79 (3)	1.71 (9)
GSI AVG 8	0.75 (3)	0.54 (2)	1.32 (7)
GSI AVG 16	0.54 (2)	0.52 (2)	0.93 (4)
GSI AVG 32	0.41 (2)	0.42 (1)	0.71 (4)
GSI AVG 64	0.17 (2)	0.3 (1)	0.5 (4)
GRAD ROB	1.87 (8)	1.64 (66)	3.75 (16)
MEAN 3	0.87 (4)	0.71 (3)	1.67 (9)
MEAN 5	0.57 (2)	0.53 (2)	1.14 (6)
MEAN 7	0.44 (2)	0.36 (2)	0.73 (4)
MEAN 9	0.38 (2)	0.43 (1)	0.7 (3)
VAR 3	2.46 (12)	1.44 (6)	40.95 (190)
VAR 5	1.65 (10)	1.11 (5)	34.85 (181)
VAR 7	1.31 (8)	0.96 (4)	28.94 (135)
VAR 9	1.09 (5)	0.95 (6)	22.28 (125)
VARIATION OF THE PIXELS WITHOUT OBJECT			
INTENSITY 145	1.56 (9)	1.25 (5)	1.45 (7)
INTENSITY 4	2.21 (11)	2.12 (9)	2.12 (9)

GSI	Gray scale image
GSI AVG N	Gray scale image averaged N times
GRAD ROB	Gradient image using Roberts operator
MEAN N	Local mean with window size N*N
VAR N	Local variance with window size N*N

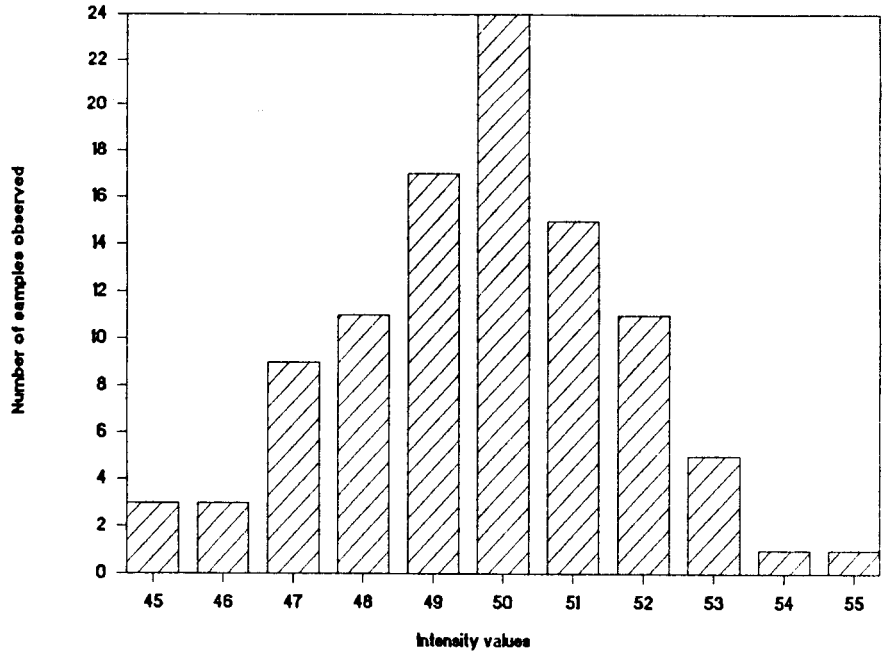


Figure 3. Intensity distribution of one pixel due to noise.

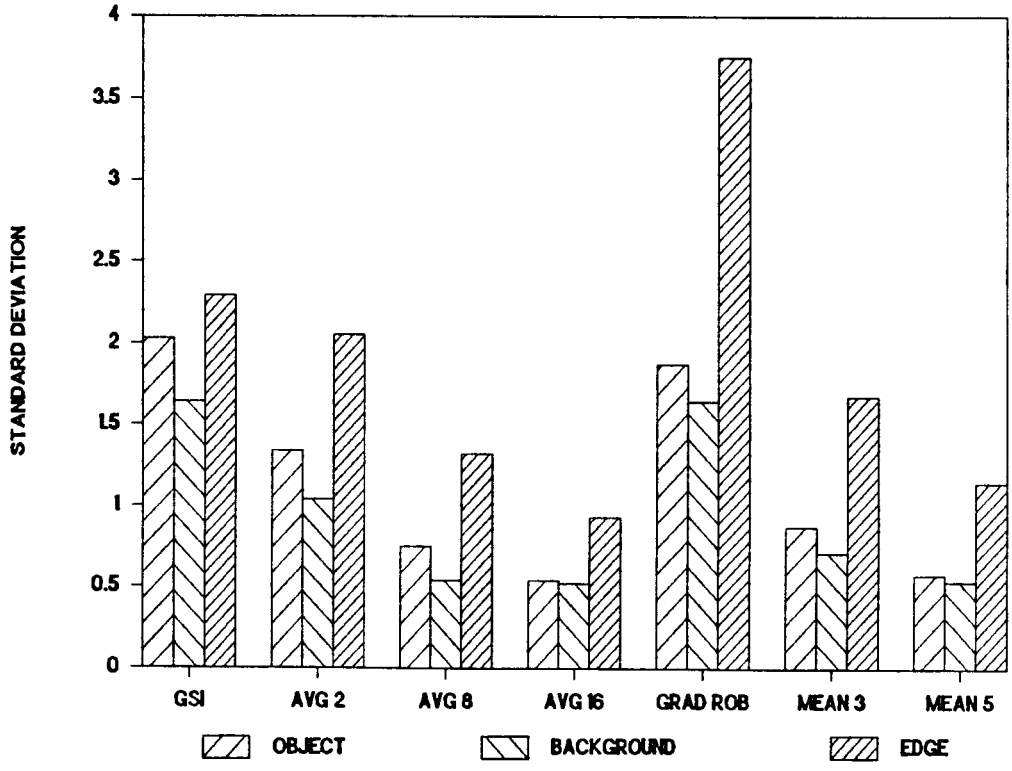


Figure 4. Intensity variation for 100 samples - comparison of techniques

2.3 DIMENSIONAL INSPECTION WITH VISION SYSTEMS

The process of dimensional inspection with vision systems is divided into several steps. First, a gray-scale image is captured from a part to be inspected. Image processing methods then may improve the quality of the image by smoothing operations or noise filtering.

The next step is the process of edge detection. Depending on the method used, the decision whether or not a detected pixel might belong to an edge has to be done by thresholding. Displaying the enhanced edge pixels in a binary space results in a contour image of the part. In this stage, information about an edge is stored in the form of a set of x and y coordinates of edgepoints in the digital coordinate system of the frame buffer.

For example, by the use of least square methods, the edge pixels, expressed in x,y coordinates, are transformed into a geometrical representation of a circle.

The result of the dimensional inspection process is subject to errors. The errors are distinguished between systematic and statistical errors. Systematic errors are characterized by an assignable cause. The act of calibration described in Chapter IV compensates for such errors. Common systematic errors occur due to light variation or optical errors. Image noise in random form introduces statistical errors. With the help of different image analysis methods it is possible to reduce noise of an image and to extract the edge of an object to be measured.

2.4 IMAGE PROCESSING AND ANALYSIS METHODS

Rueff and Melchior [11] have developed an approach to organize and structure image analysis methods. The methods were divided into orthogonal and nonlinear transformations, point, local, and global operations, which are related in a mathematical framework (Figure 5.).

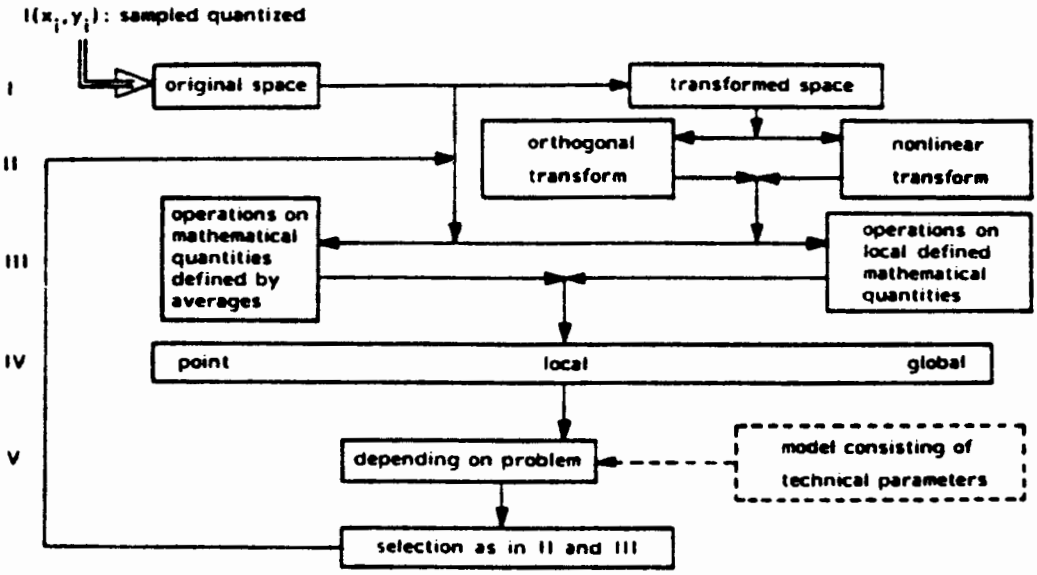


Figure 5. Mathematical framework of image analysis methods [11].

In this project, point, local, and global operations are investigated. Image processing in transformed space could be the subject of later research.

The following operations were studied and implemented in this project.

Point Operations

Threshold (a)

Threshold (b) - Highcut, Lowcut

Image Averaging

Local Operations

Variance Mask (3)

Variance Mask (5)

Roberts Edge Filter

Filtering of Image Noise using Local Statistics - Lee

Global Operations

Contour Following

Automatic Threshold Selection

Arc Center and Radius Estimation

2.4.1 Point Operations

Point operations are mathematical operations on one pixel.

Threshold (a). Threshold (a) creates a binary image by using the following operation.

$$N(x,y) \in F \quad \text{intensity}(x,y) = \begin{cases} 0, & \text{intensity}(x,y) < \text{threshold} \\ 255, & \text{intensity}(x,y) \geq \text{threshold} \end{cases}$$

where F is the set of pixels stored in the Frame Buffer,

and intensity(x,y) $\in$ {0..255}.

Threshold (b) - Lowcut, Highcut. Threshold (b) creates a binary image by using the following operation.

$$\forall\ (x,y) \in \Gamma \qquad \text{intensity}(x,y) = \begin{cases} 0, & \text{intensity}(x,y) < \text{lowcut} \\ 255, & \text{lowcut} \le \text{intensity}(x,y) \le \text{highcut} \\ 0, & \text{intensity}(x,y) > \text{highcut} \end{cases}$$

Threshold operations are chosen because they represent a classical method in the use of vision systems for dimensional inspection. The advantage of thresholding, in general, is that the transformation into a binary image is done by hardware, and therefore can be performed very fast.

Thresholding (b) is a variant of method (a) by using high and lowcut threshold values and incorporates the advantage that the thickness of the contour, in other words the number of edge pixels, can be controlled.

Image Averaging. Image averaging of static images results in the reduction of the normal distributed random noise [14]. The intensity values of several images are combined pixel by pixel and the mean is calculated. This simple image preprocessing method is chosen because of its capability to reduce noise without using local operators. Image averaging can be performed by the image processing software.

2.4.2 Local Operations

Local operations are mathematical operations on one pixel incorporating spatial relations.

Variance (3)(5). Variance (n) calculates the variance of the intensity of a pixel in an n x n neighborhood as:

$$\text{variance (n)} = \sum_{i=\frac{-(n-1)}{2}}^{\frac{n-1}{2}} \sum_{j=\frac{-(n-1)}{2}}^{\frac{n-1}{2}} \frac{\text{intensity}(x+i,y+j) - \text{mean}(x,y)}{n^2}$$

for $n \in \{3,5,7,..\}$, and mean(x,y) the average intensity value of the nxn neigborhood.

The magnitude of the variance is a measure of the strength of an edge. Since an edge is a high contrast area, the magnitude of the variance of an edge pixel is high. The edge is defined as the set of pixels in which the variance value exceeds a given threshold.

Roberts Edge Filter

Roberts cross point edge filter belongs to the methods of gradient operators. Roberts [15] defines the magnitude of the gradient as the maximum of the differences of the intensity values between two crossed adjacent pairs of pixels (Figure 6). The Roberts gradient value for a pixel is defined as:

$$\max \left\{ \left| i(x,y) - i(x+1,y+1) \right| , \left| i(x,y+1) - i(x+1,y) \right| \right\}$$

where the function i(x,y) is the intensity(x,y) of a pixel.

This well known edge detection method is chosen because in Peli and Malah's [9] comparison of edge detection algorithms, its performance was shown to be good on ramp edges, and on ideal edges superpositioned with Gaussian noise, for high signal to noise ratios. As in the previous method, the edge is determined by pixels with their gradient

value exceeding a specified threshold.

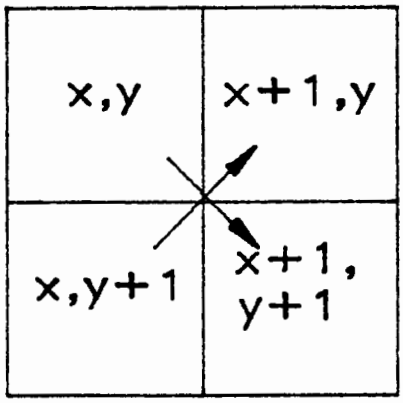


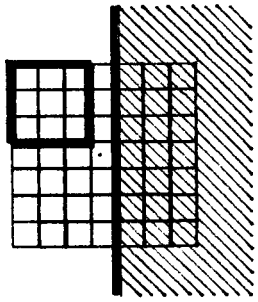
Figure 6. Roberts edge filter

Filtering of Image Noise Using Local Statistics

Lee [16][17] presented an algorithm for noise filtering by calculating mean and variance of a reduced subset of pixels incorporating gradient information (Figure 7). For every pixel, the intensity information of its 7 x 7 neighborhood is reduced to a 3 x 3 window by calculating the submeans of the 9 overlapping subareas of size 3 x 3 pixels. Within this subarea, the three-level simple mask of Robinson [18] is applied. A mask operation on a pixel intensity value is performed by replacing the intensity for a pixel with the new value

$$\begin{aligned} inew(x,y) = {} & w11\ i(x-1,y-1) + w12\ i(x-1,y) + w13\ i(x-1,y+1) + \\ & + w21\ i(x\ \ ,y-1) + w22\ i(x\ \ ,y) + w23\ i(x\ \ ,y+1) + \\ & + w31\ i(x+1,y-1) + w32\ i(x+1,y) + w33\ i(x+1,y+1) \end{aligned}$$

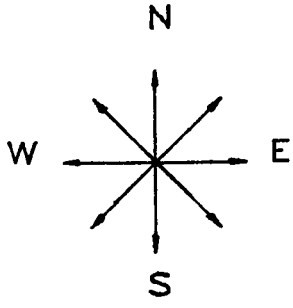
where w11 - w33 are the values of the matrix of the mask.



7 x 7 window for pixel z(i,j)

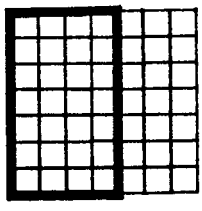
m11	m12	m13
m21	m22	m23
m31	m32	m33

Reduction to a 3 x 3 matrix by calculating means of 3 x 3 sub matrices



Calculation of gradient directions

Calculation of mean and variance of submatrix



Replacement of pixel z(i,j) with xnew = f(mean,variance,z)

Figure 7. Filtering of Image Noise using Local Statistic

Example of Three-Level Simple Masks:

$$\text{North} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\text{Northwest} \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

The Three-Level Simple mask detects edges in a specific direction, for example north, by calculating the gradient. Based on the gradient information obtained in the previous step, the subset which is the most likely to represent the center pixel of the 7 x 7 window is chosen. The mean and the variance are calculated from this subset. The center pixel is replaced by the new value calculated by

$$\hat{x}_{i,j} = \bar{x}_{i,j} + \frac{Q_{i,j}}{Q_{i,j} + \sigma_1^2} \cdot \left(z_{i,j} - \bar{x}_{i,j} \right)$$

where $\bar{x}_{i,j}$ is the mean of the subset of the 7 x 7 window;

$Q_{i,j}$ is the variance of the subset of the 7 x 7 window;

$z_{i,j}$ is the original intensity value of the center pixel;

σ_1^2 is the assumed noise variance.

For a low contrast area $x_{i,j} = \bar{x}_{i,j}$ and for a high contrast area for example an edge $x_{i,j}=z_{i,j}$

This algorithm is chosen because it represents the group of local operators with noise filtering capability by preprocesssing the image. The source code of the algorithm is included in Appendix A.

2.4.3 Global Operations

Contour Following. A contour following algorithm is capable of extracting closed contours in binary images of an object. Due to the built-in backtracking algorithm, it is possible to follow the contour of sharp or noisy edges. The essence of this algorithm is given below. The contour search process can be divided into four steps:

1. A window has to be specified that contains the contour to be formed
2. The initial edge pixel of the object is searched in the first quadrant of the window. The coordinates are stored on the first position of an x,y data array with length 0 .. maxlength.
3. The next boundary pixel is found by searching a 3 x 3 neighborhood for a pixel which must satisfy the following conditions:
   - the intensity of the pixel $I(i_x, j_y)$ is equal to the intensity of the object;
   - the degree of exposure (DOE) (Figure 8) is:
     $\max[DOE(i_x, j_y)]$, $\forall (i_x, j_y) \in S \wedge 0 < DOE(i_x, i_y) < 4$
   - the pixel is not already found within the last n boundary points.

   The degree of exposure (DOE) of a pixel i_x, j_y is defined as the number of adjacent pixels with the intensity of the background. Where adjacent pixels of (x,y) are specified as the set of points

$S = \{(x+1,y);(x-1,y);(x,y+1);(x,y-1)\}$

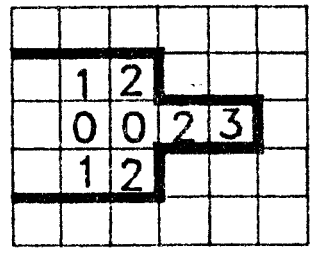


Figure 8. Degree of Exposure - The object pixel with number 3 is surrounded by 3 background pixels and therefore DOE(x,y)=3 for this pixel.

4. The new boundary pixel is stored in the array of x,y coordinates and the next boundary pixel has to be found. The condition to end the search is when the new boundary pixel is equal to the initial pixel or when the length of the data array exceeds the maximum specified length.

The following situations will require exceptional handling:

1. If the degree of exposure of the last boundary pixel in the array of edgepixels is DOE = 3, and all other pixels within the neighborhood are either already stored edge pixels or pixels with background intensity, the algorithm would not be capable of finding a new boundary pixel. Therefore a backtracking algorithm is incorporated. Backtracking: If no new boundary pixel is found, the backtracking algorithm searches within the last n boundary points for the next boundary pixel. For this task, exactly the same procedures are used as in the forward tracking. The new boundary pixel found is stored at the i+1th position where the backtracking mechanism started. Another possibility would be to

skip the boundary points which were backtracked and to search on from the point where the new boundary pixel was found. This would result in smoothing the contour of the object.

2. Another problem occurs if in the search process for the initial pixel, pixels with object intensity were found which do not belong to the object contour. Such situations occur with increasing noise, exceeding the threshold. In this case, the algorithm continues searching for an initial boundary pixel of the object, or fails if the first quadrant is completely searched. The source code of this algorithm is given in Appendix B.

Automated Threshold Selection

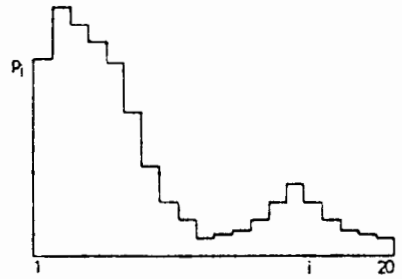
To automate the task of thresholding gray level images into binary images, an algorithm proposed by Johannsen and Bille [19] was implemented (Figure 9).

By computing the Shannon's entropy of the gray level diagram, the gray levels are separated into two subsets, G_k and G_{k-1} so that the interdependence between them is minimized. The entropy of a distribution is defined as:

$$H = -\sum_{k=1}^{n} p_k \cdot \ln p_k \qquad \text{and } p_k = \frac{N_k}{N}$$

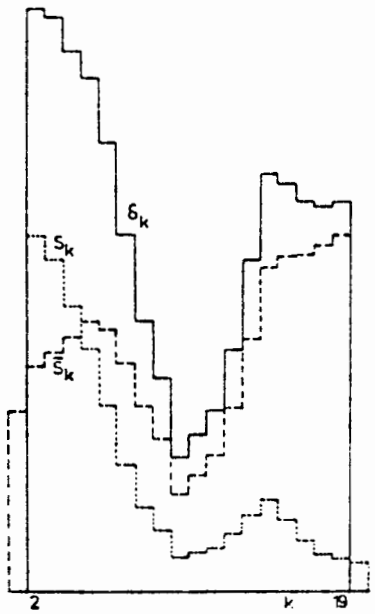
where pk is the distribution of the intensity, Nk the number of pixels with the intensity k in the image and $N = \Sigma\, Nk$.

Probability Distribution



divide in subset Gk, $\bar{G}k$ for all k

calculate Sk, $\bar{S}k$ with S as a measure of the contribution of gray level k to the part of the histogram composed of the divided subsets Gk and $\bar{G}k-1$



Min(delta k)=Min(Sk+$\bar{S}k$)
Threshold value = k

Figure 9. Automated threshold selection

By minimizing the sum of the entropies S_k and $\bar{S}_k$ of the divided subsets, where S_k and $\bar{S}_k$ measure the contribution of a gray level K to the part of the histogram composed of the divided subset G_k and G_{k-1}, the threshold value K can be determined by

$$\min(\delta_k) = \min(S_k + \bar{S}_k),$$

where the threshold = k.

This algorithm supports thresholding (a). The source code of the implemented program is included in Appendix C.

Arc Center and Radius Estimation

To transfer edge information into geometric representation, the pixel coordinates have to be processed by an estimation algorithm. In the case of circular features, the geometry of the object is described in diameter and x,y coordinates of the center of the circle. The algorithm used for this purpose has been proposed by Thomas and Chan [20].

The error between a constant area $\pi \cdot R^2$ and the area of the circle centered at $\bar{x},\bar{y}$ is minimized. This results in the equation

$$e(R,\bar{x},\bar{y}) = \sum_{i=1}^{N} \left[\pi R^2 - \pi \cdot \left\{ \left(x_i - \bar{x}\right)^2 + \left(y_i - \bar{y}\right)^2 \right\} \right]^2$$

in which its partial derivatives are set to zero to find the minimum.

$$\frac{\partial e}{\partial R} = 0 \qquad \frac{\partial e}{\partial \bar{x}} = 0 \qquad \frac{\partial e}{\partial \bar{y}} = 0$$

This results into two linear equations. Using standard algebraic calculations handling summation terms and solving the two linear equations for the two unknowns $\bar{x},\bar{y}$, the center of the estimated circle is calculated. Therefore, the radius yields

$$R^2 = \frac{1}{N}\left\{ \sum x_i^2 - 2\bar{x} \sum x_i + N\bar{x}^2 + \sum y_i^2 - 2y \sum y_i + N\bar{y}^2 \right\}$$

The source code is documented in Appendix D.

2.5 SETUP OF METHODS

For the process of dimensional inspection, the image operations, available in the form of software subroutines, are linked together. The point, local, and global operations are combined to evaluation algorithms, starting with the acquisition of an image and ending with the results of the measurement. The output image of an evaluation algorithm is a binary contour image of the object. At this stage, edge information is available in the form of x,y coordinates of the frame buffer. By using estimation and least square methods, edge information is transformed into a geometric representation of the object. In the process of measurement of circular features, the x,y coordinates are parametrized in terms of diameter and the center coordinates of the circle (Figure 10). The Figure 11 - 15 show the structure of the different evaluation algorithms, implemented for the experiments in this work. To avoid confusion, in the rest of this document the expression "method X" refers to the evaluation algorithm using the point or local "image analysis operation X".

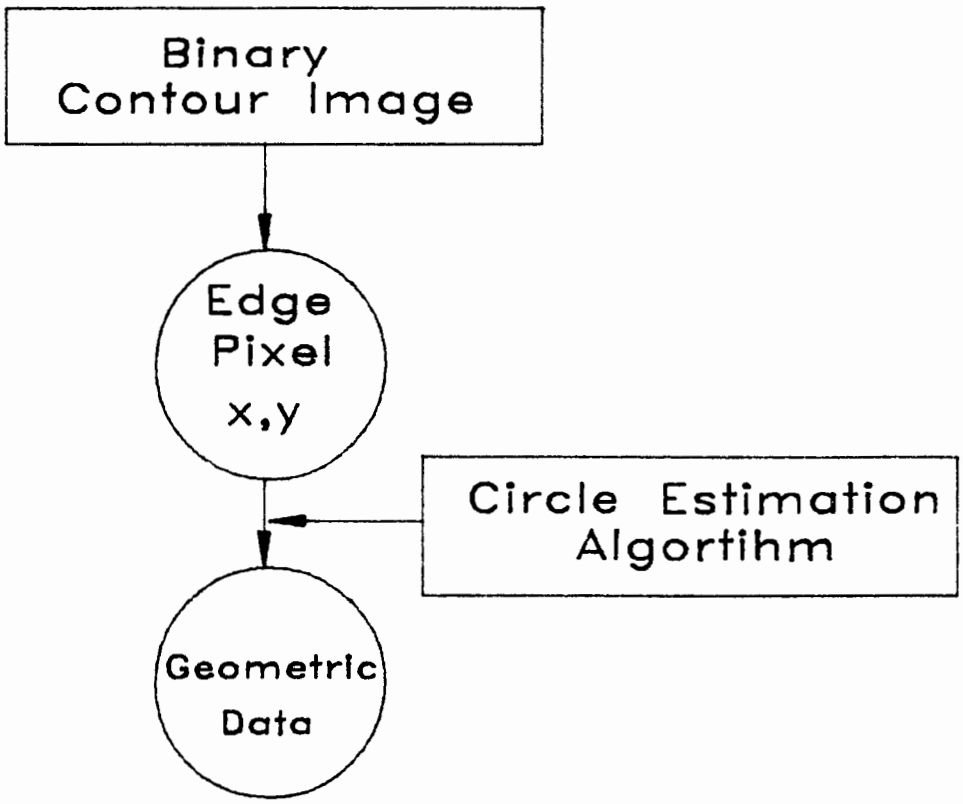


Figure 10. Transformation into geometric space

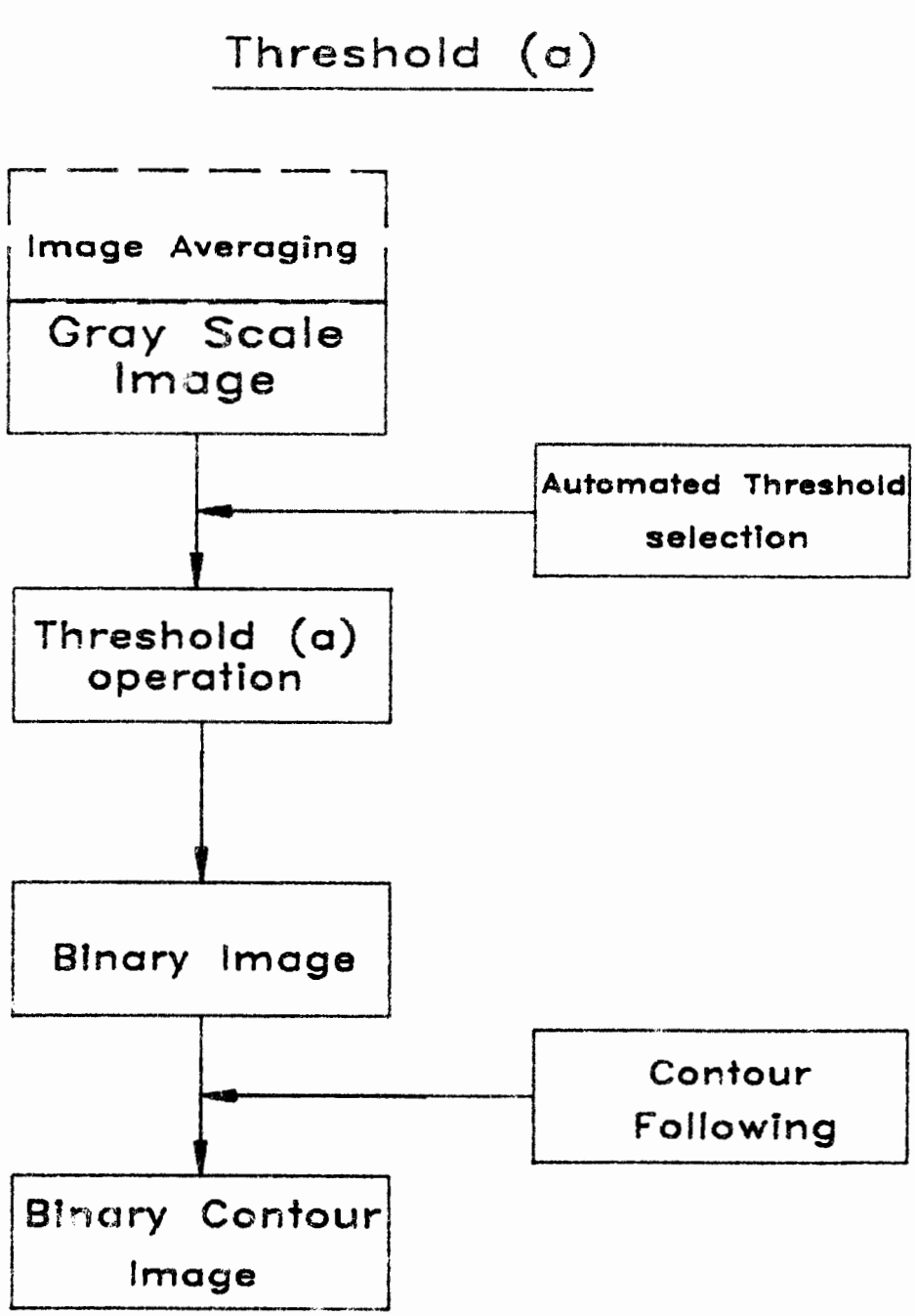


Figure 11. Threshold(a) method. The evaluation program acquires a gray scale image, including the option of image averaging. The automated threshold selection operation determines the threshold. By using the threshold(a) operation, a binary image is obtained. The contour following algorithm extracts the edge pixels, which are displayed as a binary contour image.

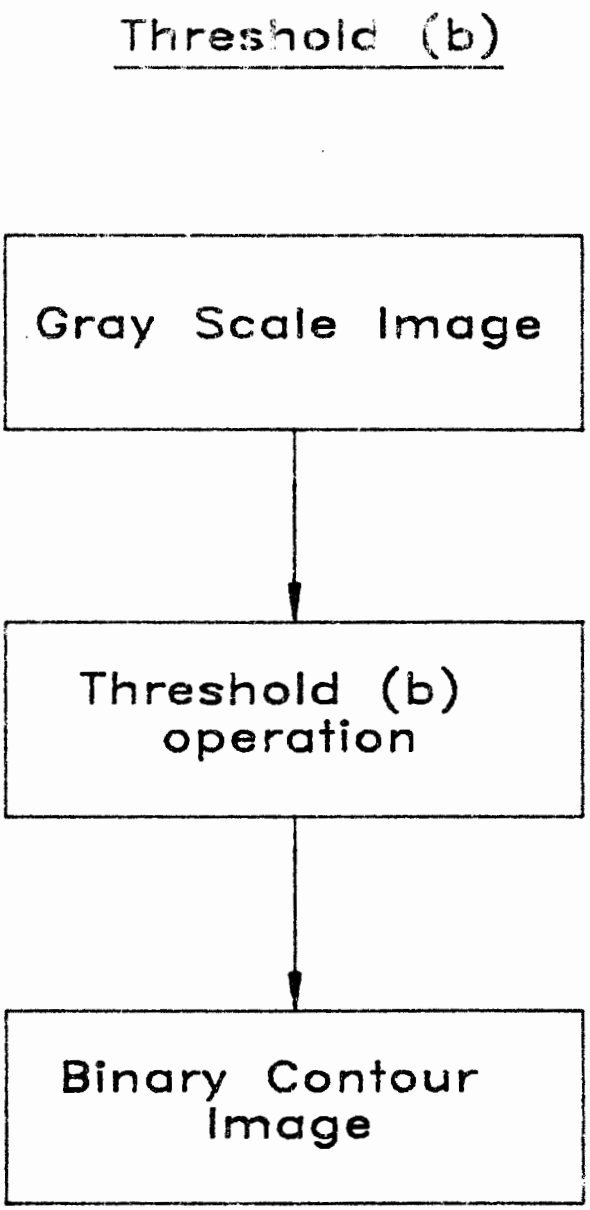


Figure 12. Threshold(b) method. The acquired gray scaled image is transformed into a binary contour image. Edge pixels are pixels with intensity: lowcut < intensity < highcut.

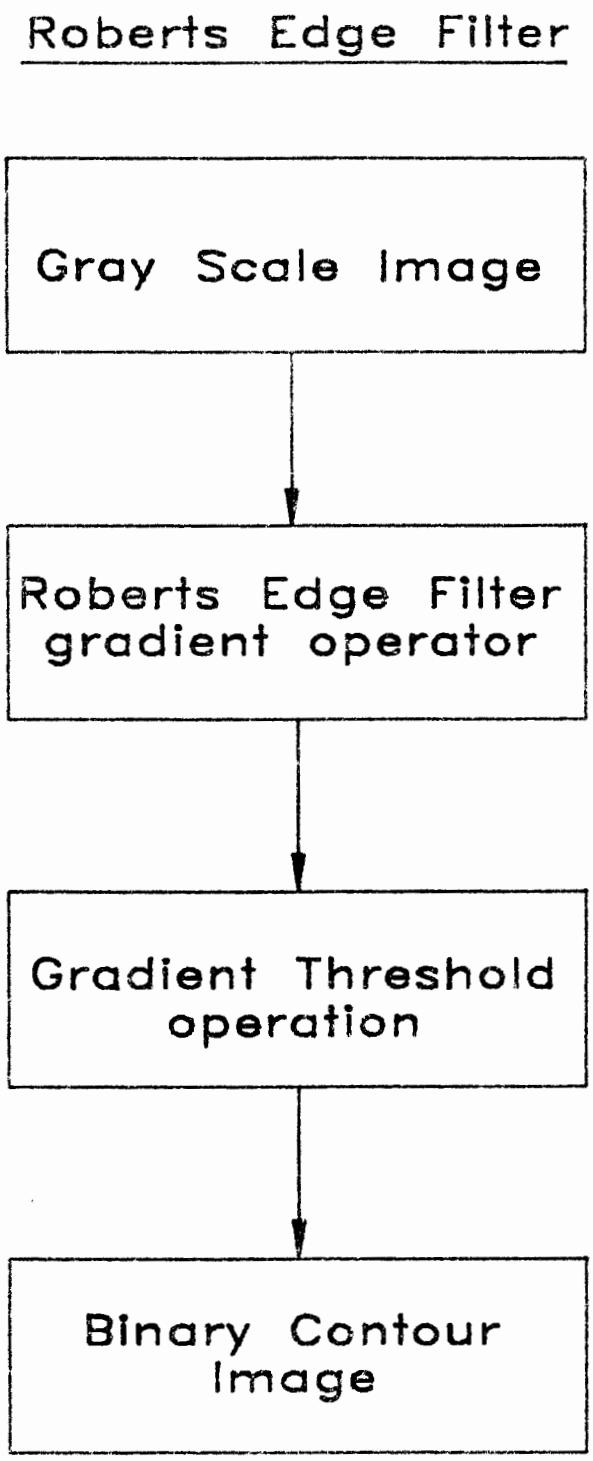


Figure 13. Roberts edge filter method. The gray scale image is converted into a gradient image. Gradient values exceeding a specified threshold form the edge and are displayed in the binary contour image.

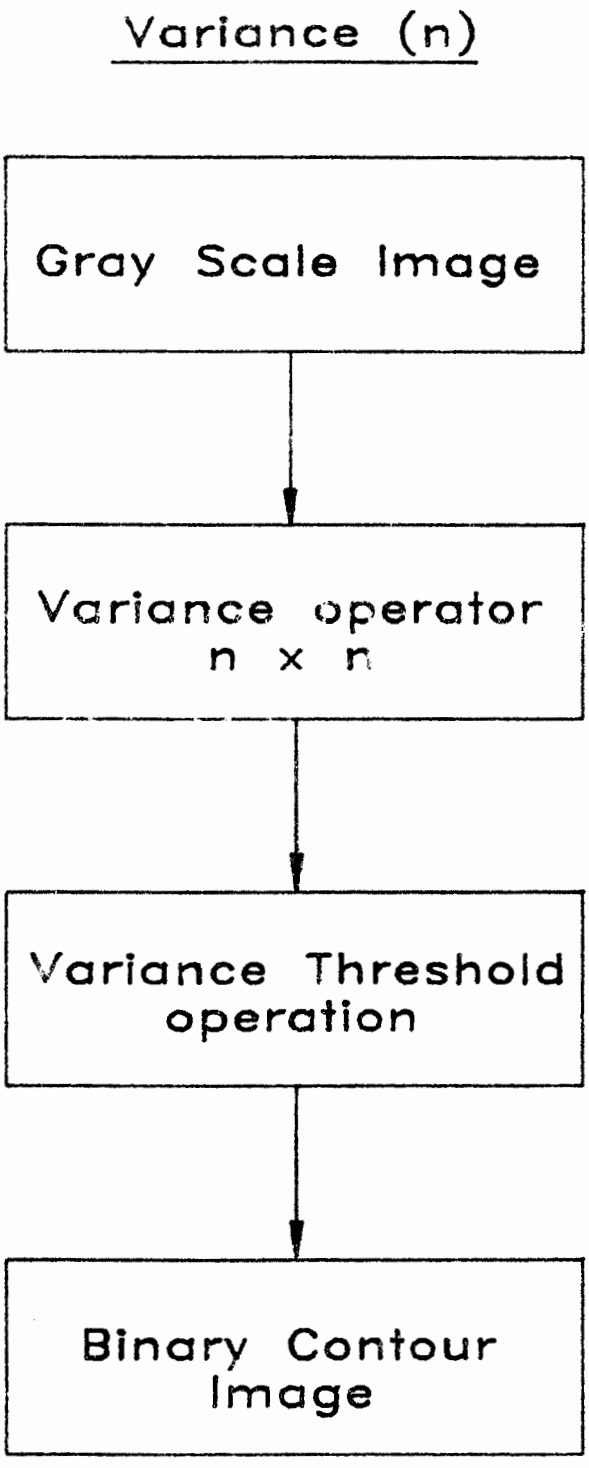


Figure 14. Variance(n) method. Similar to Roberts edge filter, the edge pixels of variance (n) are pixels exceeding a specified variance threshold.

Filtering of image noise using Lee's Local Statistic method

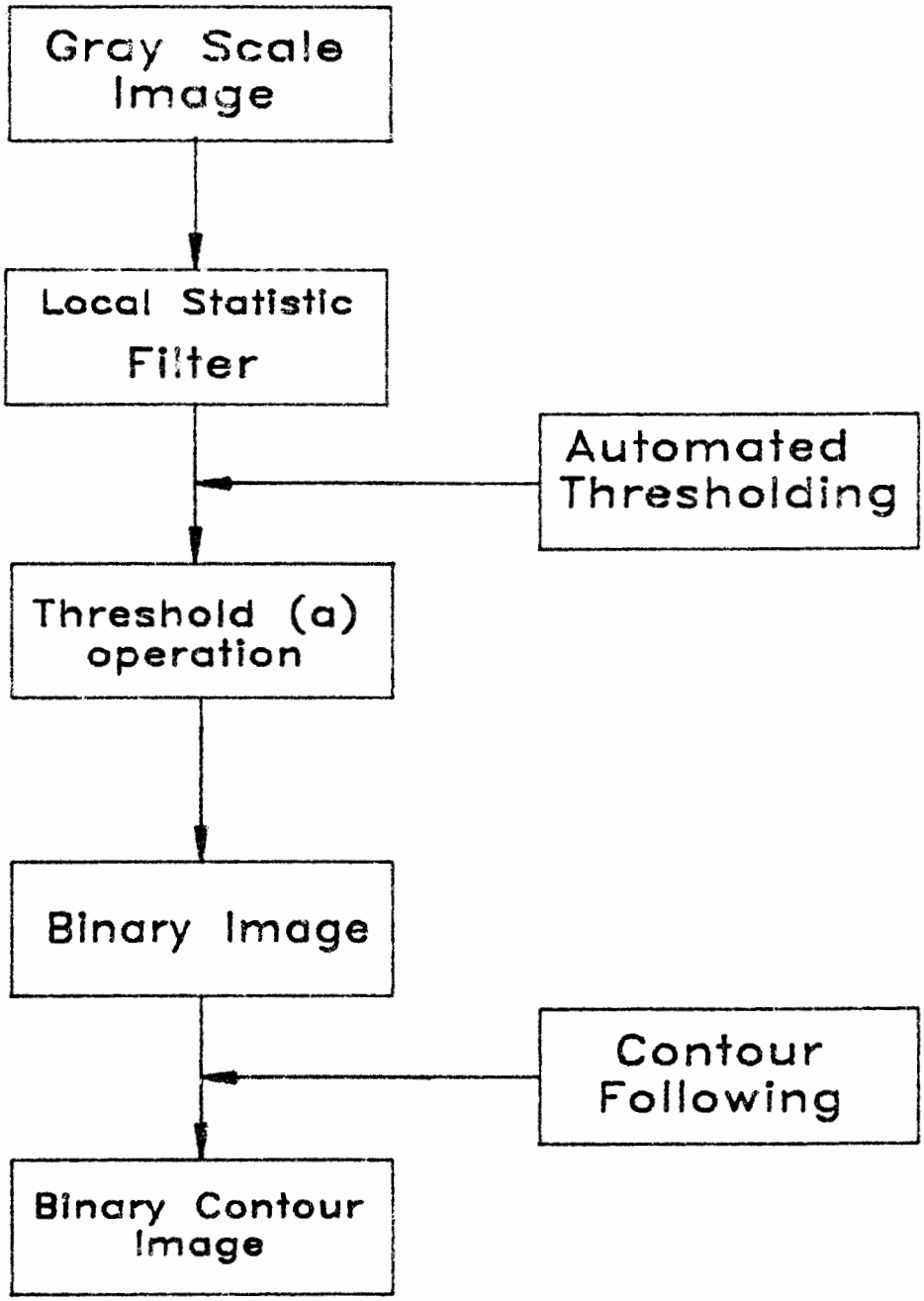


Figure 15. Local Statistic method. The gray scale image is processed by the local statistic filter. Based on the filtered gray scale image, the same operations are performed as in the threshold(a) method.

Since the local methods variance(n) and Lee's Local Statistic Method take into account a large set of neighboring pixels, these operations should be applied only on those pixels which represent the edge or are close to it. This can be done in two different ways. Both alternatives were implemented and tested.

Option 1: A lower and upper intensity boundary is specified (Figure 16). Each pixel within this range is processed. The intensity range must be selected wide enough so that its boundaries do not influence the decision whether a pixel belongs to an edge or not.

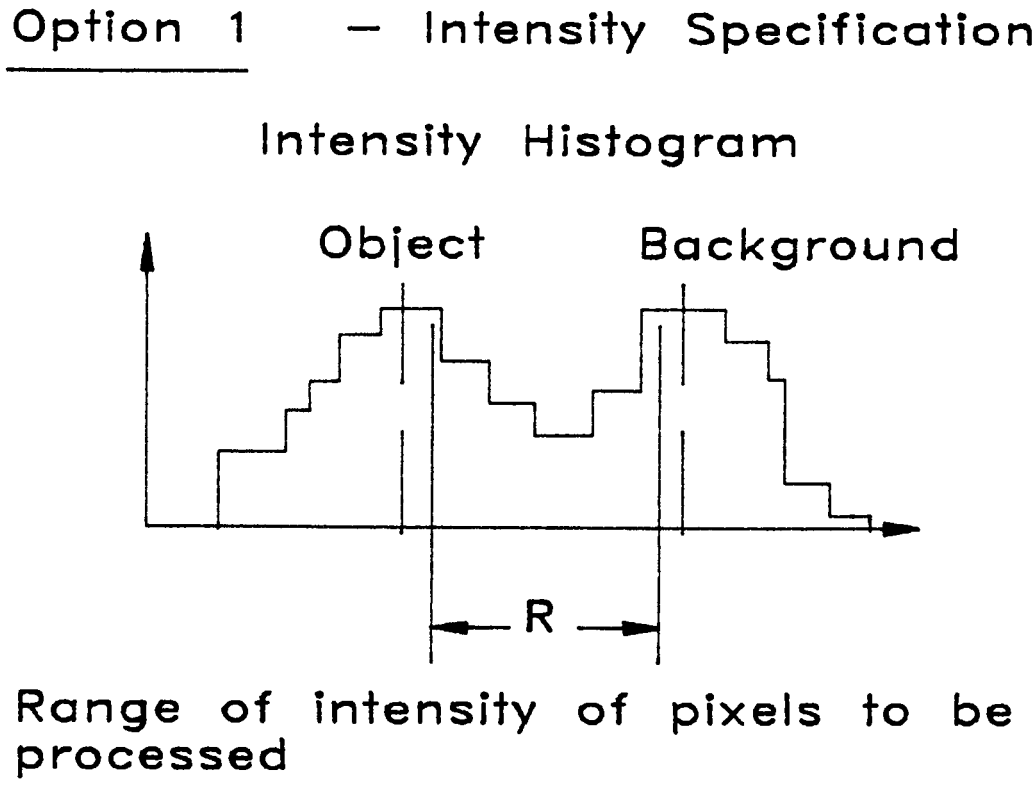


Figure 16. Option 1 of local methods

Option 2: The second possibility requires a priori knowledge of the approximate edge location (Figure 17). All pixels within a certain band of the approximate edgeline are processed. As in the previous case, this band must be wide enough so that the specification of the estimated location of the edge does not influence the selection of edgepixels.

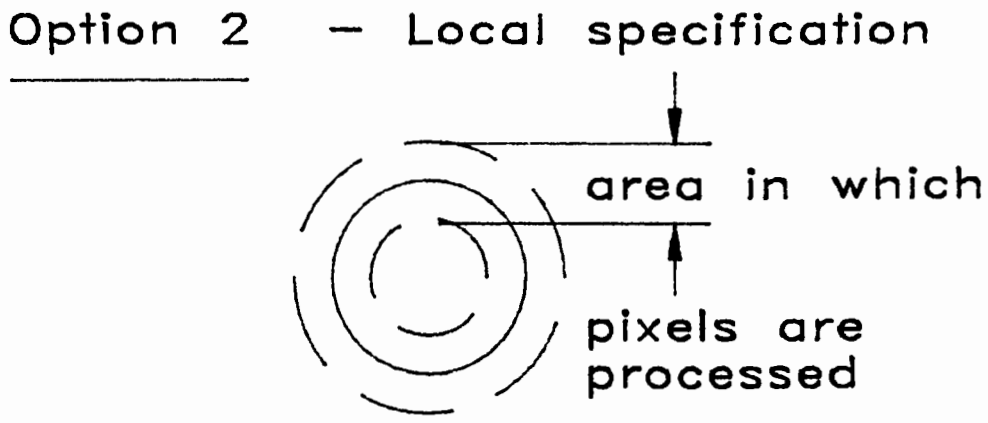


Figure 17. Option 2 of local methods

CHAPTER III

COMPARISON OF IMAGE ANALYSIS METHODS

3.1 MEASURES OF PERFORMANCE

A comparison of image analysis methods has to be based on objective and meaningful measures of performance. They can be divided into quantitative and qualitative measures.

3.1.1 Quantitative Measures

Peli and Malah [9] and also Vliet, et al [10] used Pratt's figure of merit [21] which is the weighted and normalized deviation of an actual edge point from the ideal edge.

In this project, another approach is taken. The performance of the methods is studied with regard to the geometrical parameters of a circle. The deviation of an actual diameter from the ideal diameter is the measure of geometric stability. The number of pixels used to form the circle is the measure of pixel stability and finally the repeatability of a method is the measure of the standard deviation of the diameter of n detected circles of the same scene. The advantage in using these quantitative measures is that the influence of a specific method on the obtained geometrical parameter, for example, the diameter, is directly revealed.

The above-mentioned quantitative performance measures are defined as:

1. Geometric stability (GS)

$$GS_{noise} = \frac{\left| \bar{d}_{noise} - D \right|}{D} \cdot 100$$

where the subscript "noise" specifies the SNR, $\bar{d}_{noise}$ is the average diameter obtained of n sample measurements, and D is the diameter detected by the method of an image with no noise. For a perfect image analysis method with regard to geometric stability, GS=0.

2. Pixel Stability (PS)

The number of edge pixel detected compared to the ideal edge is expressed in the measure of pixel stability as:

$$PS = \frac{\text{number of ideal edge points}}{\text{number of detected edge points}}$$

Pixel stability is the inverted measure of the mean width of an edge [9], which is a known measure in image processing. The inversion is done to obtain a better graphical representation of the results. For a perfect image analysis method with regard to pixel stability, PS=1.

3. Repeatability of Measurement

The standard deviation (STD) of the diameter is defined as a measure of repeatability of a method as follows.

$$STD = \left(\frac{\sum (d - \bar{d})^2}{n} \right)^{\frac{1}{2}}$$

where d is the diameter of one of n detected circles and $\bar{d}$ is the average diameter of the n samples. For a perfect image analysis method with regard to repeatability, STD=0.

3.1.2 Qualitative Measures

The purpose of the qualitative measures is to give an idea of the appearance of the binary contour image obtained by each method. It is compared to the quantitative measures subjected to the human judgement capability. To assure equal comparison, the following criteria are established.

Type of contour: CL - closed contour 100%
B - broken contour, contour < 80%
CL,B - almost closed, contour > 80% but broken at some points

Noise occurrence: N - noise pixels occur which do not belong to the contour

Smoothness of the circle contour: + - smooth edge
\- - rough edge

Failure: X - failure of algorithm

3.2 DECISION MODEL

Using the performance measures, a way must be found to select an optimal method for a certain type of application. For every quantitative and qualitative performance measure, a failure criteria can be defined (Figure 18). The failure of a method is determined by the application and the accuracy demanded by the process of dimensional

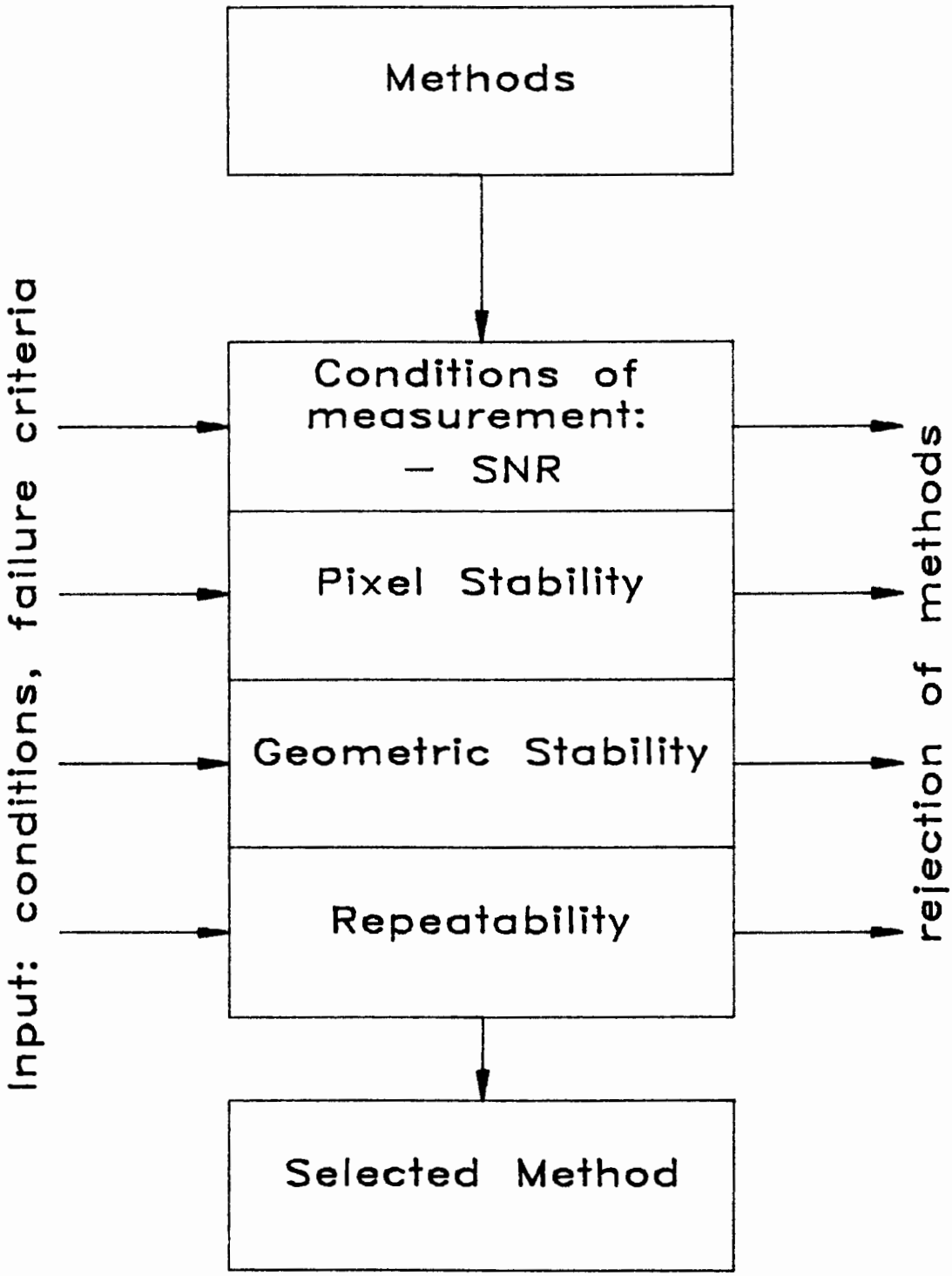


Figure 18. Decision model

inspection. Using these failure criteria a Decision Model is established. With a certain accuracy required, the failure criteria for pixel, geometric stability, and repeatability can be established. By choosing the measurement conditions, for example, the signal to noise ratio, an additional selection criteria is found. The available set of methods is analyzed whether or not the criteria of the measurement can be satisfied. Rejecting the methods failing this criteria, a subset of methods is selected which can be used for the required task of measurement.

3.3 IMAGE GENERATOR

The comparison of these methods is based on images generated by the computer. Generating artificial images rather than using the camera has the advantages that the environment, texture, and geometry of the object and the noise level can be simulated and controlled.

The image generator developed in this project consists of four modules.

Module 1 - Generates a filled circle. Object and background intensity can be specified.

Module 2 - Whereas the first module generates a step edge, the second module smooths the edge and converts it into a ramp edge of width n.

Module 3 - Normally distributed random noise with mean n and standard deviation m is added to the image.

Module 4 - Modeling of texture is done by adding normally distributed noise on object pixels. Later, this

module could be extended by simulating texture with binary noise or by using other algorithms.

3.4 SIMULATION MODEL

The methods were compared on artificially generated images. The cylindrical parts used to study the real edge appearance had a diameter of 160 pixels using typical magnifications. Therefore, the same diameter was selected for the synthetically produced circles. Two contrast levels were chosen: A low contrast simulation of 100 gray values difference (background 140, object 40) and a high contrast simulation of 200 gray values difference (background 240, 40). The low contrast images represent an illumination scene using diffused overhead lighting. The high contrast images simulate lighting typically produced using a diffused glass table where the lights are mounted below.

The simulation was done for ramp edges with a width of 5 pixels. The edge width was determined by studying images of real edges and is similar to other research done using the same edge model [9]. In Figure 19, the modeled edge and the real edge are compared. Gaussian noise with a mean of zero was added to the image. The methods were tested for σ_n=0, 2, 4, 8, 16 and 32 for both contrasts. 10 sample images for every simulation case were evaluated. Intensity values exceeding the range of 0 to 255 were clipped. In particular, this had to be done in the high contrast simulation for σ_n=16 and 32. The results of the methods were evaluated with regard to the Signal to Noise Ratio (SNR) of the synthetical image.

The range of SNR tested in the low contrast simulation is 10 to

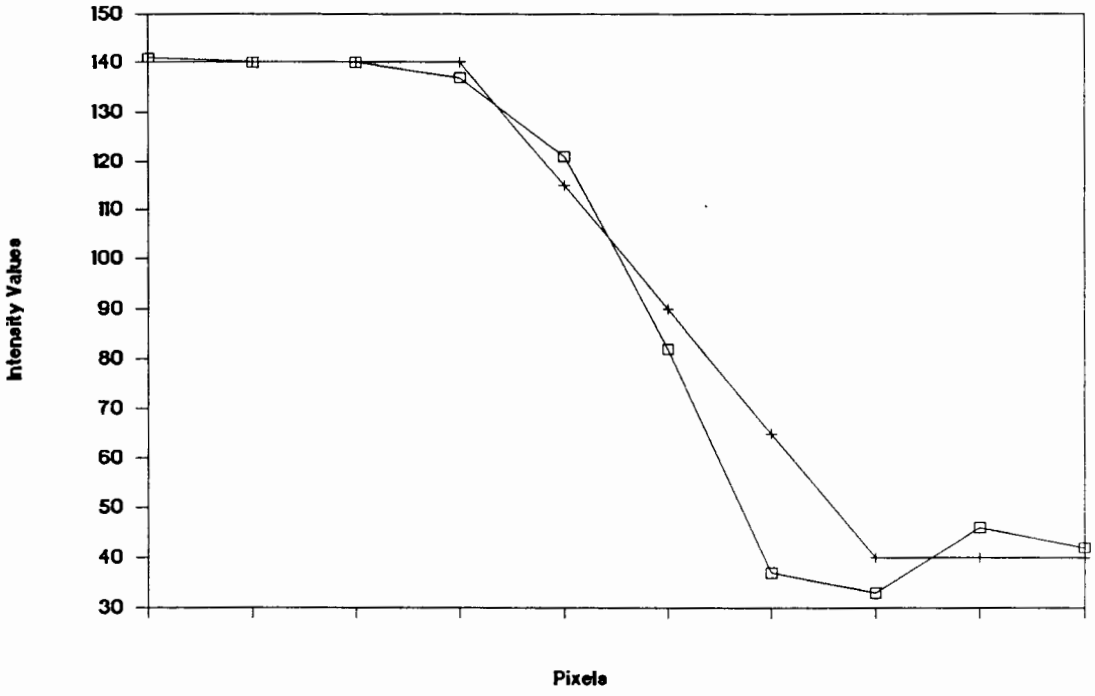


Figure 19. Simulation of edge

2500 and for the high contrast it is 40 to 10000. Although the two ranges overlap, it is advantageous to carry out both simulations, for several reasons. First, the range of SNR is extended by 10 and 10000. Second, the inaccuracy of the noise simulation due to clipping of intensity values for low SNR is revealed. Third, the redundant data may show the correctness and the weakness of the simulation model.

Every image analysis method demands a careful selection of the input parameters, for example, lowcut and highcut or the range specification on intensity for the pixels to be processed. The process of dimensional inspection requires a complete knowledge of the edge. Therefore the experiment input parameters for the methods were chosen so that for a minimum amount of computation time for each method, a closed

contour of the edge could be detected. This was done for the ideal image without noise (SNR = ∞). The input parameters for each method and the simulation case are documented in Appendix G. To have meaningful comparison of the methods, the input parameters were not readjusted for increasing noise. In addition, the experiments made during this project using these methods, showed that a readjustment of input parameters does not improve the evaluation significantly enough to take it into consideration.

3.5 SIMULATION TEST RESULTS

For the low and the high contrast simulations, the results are documented in a set of tables and figures. To familiarize the reader with the graphs, a short general description is given below for each type of table and figure. The results are analyzed and discussed in the next section.

Tables III and IV show the quantitative results of the simulation. For each method and different signal to noise ratios, the pixel stability, geometric stability and the repeatability are documented. Using data of the Tables III and IV, graphs of the pixel stability and the geometric stability versus the signal to noise ratio were generated. For each category two sets of graphs were drawn. The first set compares all methods besides the methods using evaluation option 2. The second set compares the results of the methods using options 1 and 2.

Figures 20, 21, 24, and 25 show the pixel stability versus the Signal to Noise Ratio graphs. The horizontal line for pixel stability (PS)=1 would represent a perfect image analysis method with respect to

the number of pixels used. PS > 1 means that the image analysis methods detected less edge points than the number of ideal edge points and vice versa.

Figures 22, 23, 26, and 27 show the geometric stability (GS) versus the signal to noise ratio graphs. The scale of the geometric stability is logarithmic. All methods with GS exceeding a given boundary value fail.

Tables V and VI show the qualitative results.

TABLE III

QUALITATIVE TEST RESULTS OF LOW CONTRAST SIMULATION

PIXEL STABILITY

SNR	TH(a)	TH(b)	Rob.	V3/2	V5/2	V3/1	V5/1	STA/2	STA/1
10	-	0.115	-	0.344	0.352	-	-	0.850	0.810
40	0.835	0.539	0.107	0.653	0.727	0.558	0.940	0.932	0.932
156	0.895	0.953	0.564	1.012	0.956	1.144	0.999	0.983	0.983
625	0.892	0.998	1.335	1.141	1.038	1.161	1.038	0.996	0.996
2500	0.900	1	1.300	1.110	1.053	1.110	1.053	0.998	0.998
;	1	1	1	1	1	1	1	1	1

GEOMETRIC STABILITY

SNR	TH(a)	TH(b)	Rob.	V3/2	V5/2	V3/1	V5/1	STA/2	STA/1
10	-	2.551	-	0.389	0.014	-	-	0.038	0.046
40	0.666	1.896	1.900	0.039	0.001	1.712	0.008	0.001	0.003
156	0.598	0.016	1.284	0.038	0.015	0.041	0.012	0.007	0.007
625	0.552	0.002	0.011	0.007	0.004	0.009	0.004	0.007	0.011
2500	0.441	0	0.054	0.007	0.001	0.007	0.001	0.001	0.001
;	0	0	0	0	0	0	0	0	0

REPEATABILITY OF MEASUREMENT

SNR	TH(a)	TH(b)	Rob.	V3/2	V5/2	V3/1	V5/1	STA/2	STA/1
10	-	0.428	-	0.576	0.097	-	-	0.099	0.090
40	0.064	1.050	0.279	0.086	0.035	0.749	0.033	0.065	0.063
156	0.050	0.021	1.021	0.046	0.030	0.038	0.034	0.042	0.042
625	0.046	0.004	0.048	0.035	0.021	0.032	0.021	0.035	0.035
2500	0.026	0	0.198	0.016	0.012	0.017	0.012	0.034	0.034
;	0	0	0	0	0	0	0	0	0

TABLE IV

QUANTITATIVE TEST RESULTS OF HIGH CONTRAST SIMULATION

PIXEL STABILITY

SNR	TH(a)	TH(b)	Rob.	V3/2	V5/2	V3/1	V5/1	STA/2	STA/1
40	0.837	0.595	0.174	0.841	0.871	1.052	0.929	0.961	0.960
156	0.884	0.986	0.791	1.026	0.780	1.153	0.905	0.987	0.987
625	0.891	0.999	1.368	1.091	0.927	1.112	0.927	0.995	0.995
2500	0.893	1	1.341	1.070	0.971	1.071	0.971	0.999	0.999
10000	0.910	1	1.296	1.038	1.007	1.038	1.007	1	1
;	1	1	1	1	1	1	1	1	1

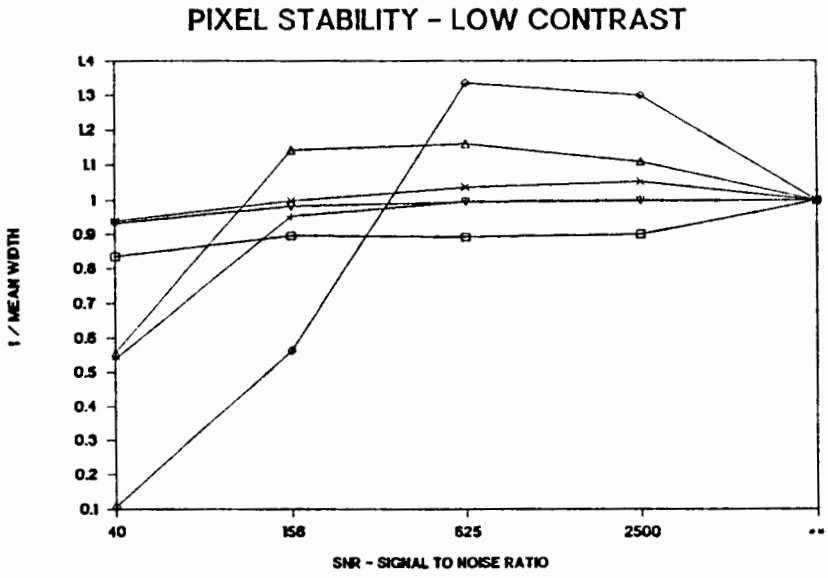
GEOMETRIC STABILITY

SNR	TH(a)	TH(b)	Rob.	V3/2	V5/2	V3/1	V5/1	STA/2	STA/1
40	0.687	1.260	6.102	0.234	0.226	2.372	0.099	0.100	0.100
156	0.638	0.008	7.305	0.042	0.105	0.009	0.096	0.032	0.032
625	0.630	0	0.008	0.014	0.030	0.010	0.030	0.002	0.002
2500	0.568	0	0.001	0.001	0.024	0.001	0.024	0.004	0.004
10000	0.437	0	0.001	0.004	0.005	0.004	0.005	0.009	0.009
;	0	0	0	0	0	0	0	0	0

REPEATABILITY OF MEASUREMENT

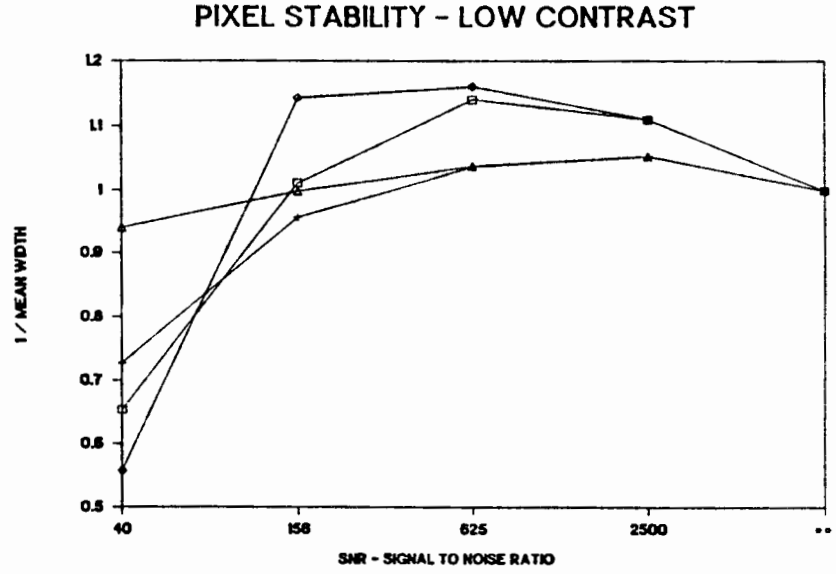
SNR	TH(a)	TH(b)	Rob.	V3/2	V5/2	V3/1	V5/1	STA/2	STA/1
40	0.058	1.102	0.599	0.018	0.058	0.673	0.058	0.068	0.066
156	0.035	0.039	0.558	0.044	0.035	0.040	0.033	0.053	0.053
625	0.041	0.003	0.015	0.032	0.032	0.023	0.032	0.045	0.045
2500	0.162	0	0.040	0.018	0.036	0.017	0.036	0.038	0.038
10000	0.041	0	0.034	0.015	0.009	0.015	0.010	0.033	0.033
;	0	0	0	0	0	0	0	0	0

- = Failure of Method
SNR = Signal to Noise Ratio
TH(a) = Threshold(a)
TH(b) = Threshold(b)
Rob. = Roberts Edge Filter
V3/2 = Variance(3) option 2
V5/2 = Variance(5) option 2
V3/1 = Variance(3) option 1
V5/1 = Variance(5) option 1
STA/2 = Local Statistics option 2
STA/1 = Local Statistics option 1



□ Threshold(a)
◇ Roberts Edge Filter
x Variance(5) option 1
\+ Threshold(b)
Δ Variance(3) option 1
▽ Local Statistic Filter

Figure 20. Pixel stability - low contrast 1



◇ Variance(3) option 1
□ Variance(3) option 2
Δ Variance(5) option 1
\+ Variance(5) option 2

Figure 21. Pixel stability - low contrast 2

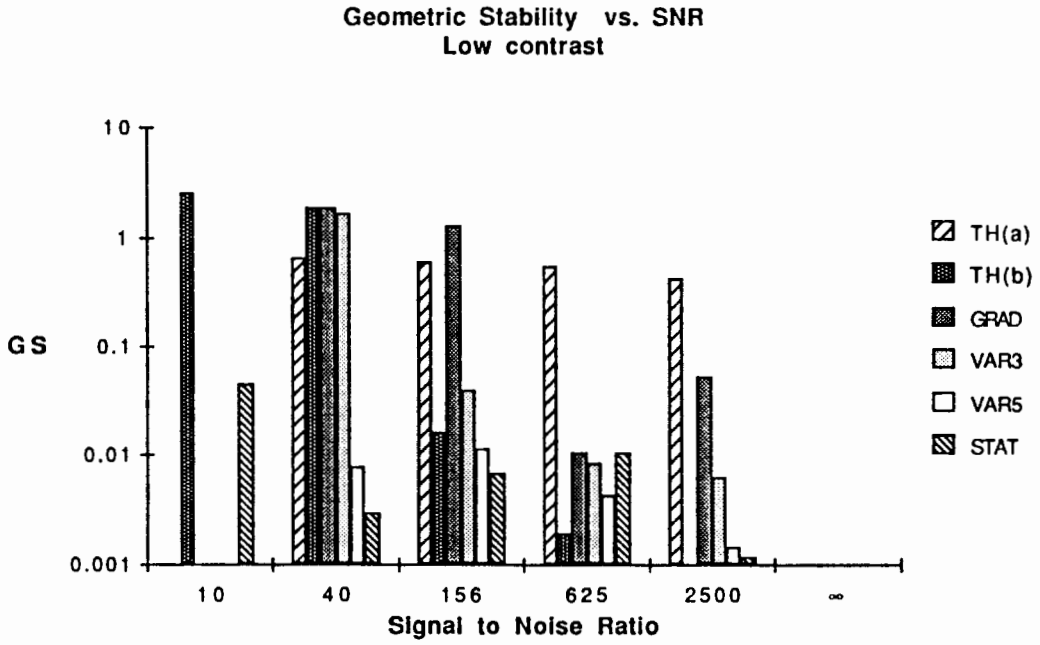


Figure 22. Geometric stability - low contrast 1

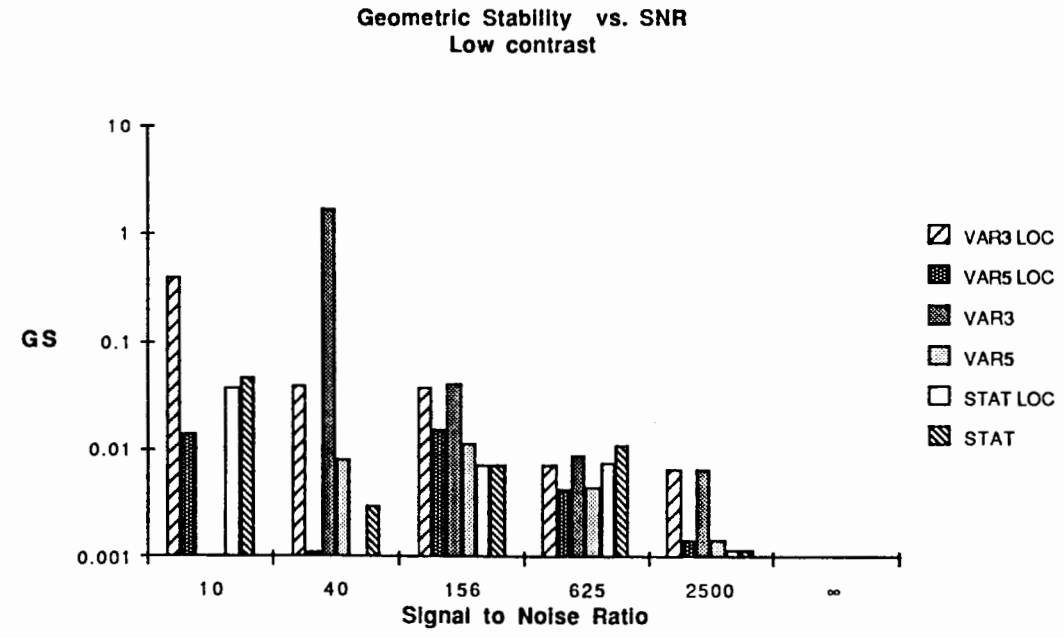
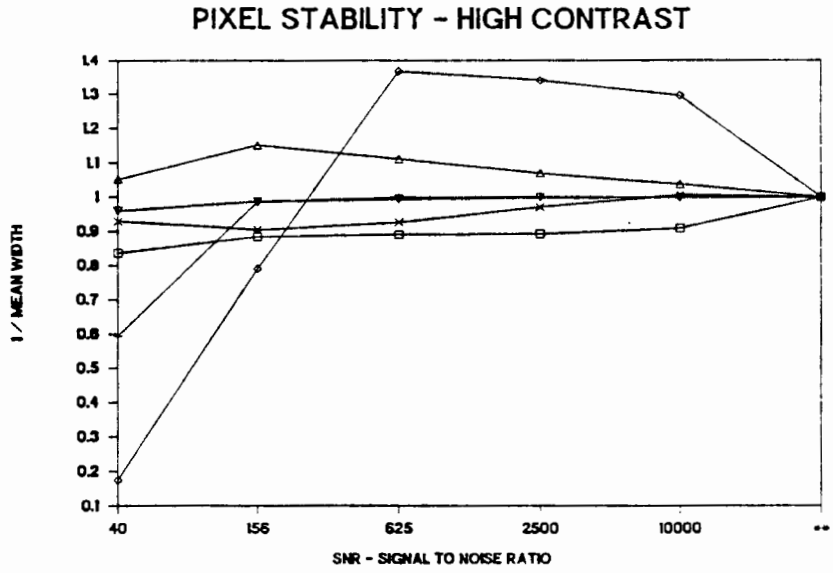
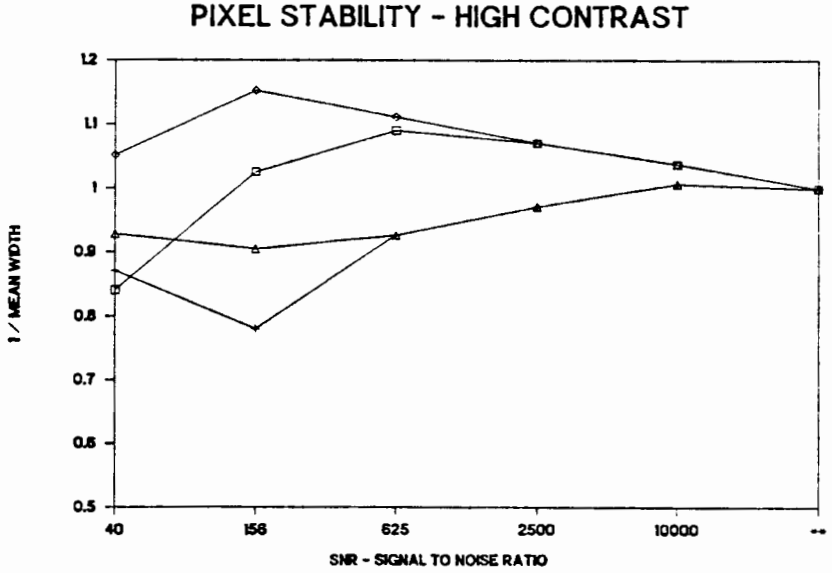


Figure 23. Geometric stability - low contrast 2



□ Threshold(a)
◇ Roberts Edge Filter
x Variance(5) option 1
\+ Threshold(b)
Δ Variance(3) option 1
∇ Local Statistic Filter

Figure 24. Pixel stability - high contrast 1



◇ Variance(3) option 1
□ Variance(3) option 2
Δ Variance(5) option 1
\+ Variance(5) option 2

Figure 25. Pixel stability - high contrast 2

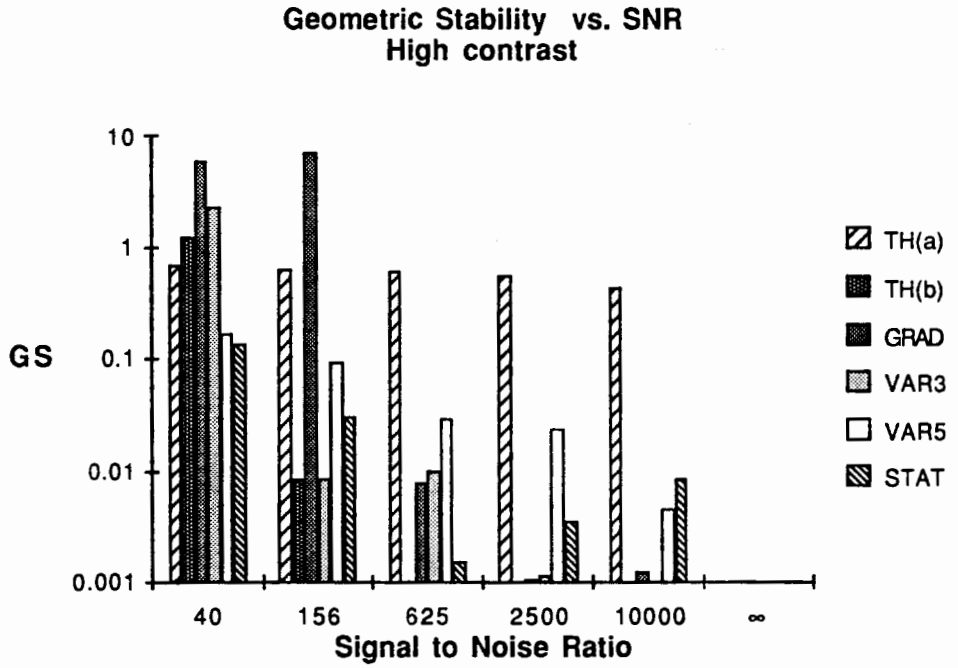


Figure 26. Geometric stability - high contrast 1

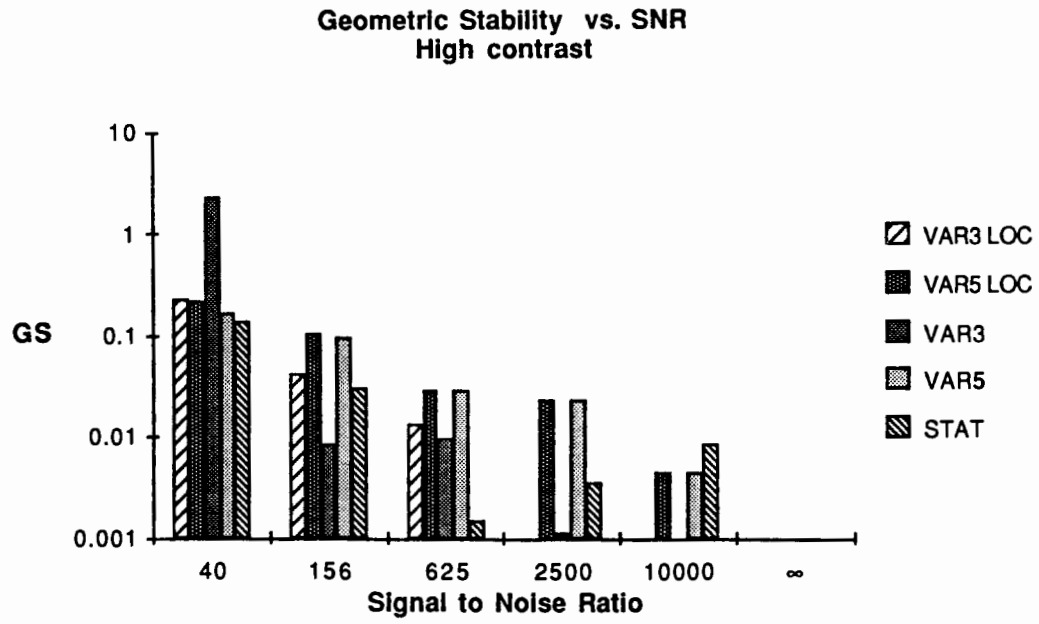


Figure 27. Geometric stability - high contrast 2

Qualitative Test Results

TABLE V

QUALITATIVE TEST RESULTS OF LOW CONTRAST SIMULATION

SNR	10	40	156	625	2500	∞
threshold(a)	X	CL-	CL-	CL	CL	CL+
threshold(b)	X,N	X,N	CL,B	CL+,B	CL+	CL+
Roberts edge filter	X	X,N	X,N	CL	CL	CL
Variance(3)	X	X,N	CL-	CL-	CL-	CL
Variance(5)	X	CL-	CL-	CL	CL+	CL+
Variance(3) opt2.	X,CL-	X,CL-	CL-	CL-	CL-	CL
Variance(5) opt2.	X,CL-	X,CL-	CL	CL	CL+	CL+
Local Statistic	CL-	CL-	CL	CL	CL+	CL+
Local Statistic opt2.	CL-	CL-	CL	CL	CL+	CL+

TABLE VI

QUALITATIVE TEST RESULTS OF HIGH CONTRAST SIMULATION

SNR	40	156	625	2500	10000	∞
threshold(a)	CL-	CL-	CL-	CL-	CL+	CL+
threshold(b)	X,B	CL,B	CL+,B	CL+,B	CL+,B	CL+
Roberts edge filter	X,N	CL,N	CL	CL	CL	CL+
Variance(3)	X,CL	CL-	CL-	CL	CL	CL
Variance(5)	CL	CL	CL	CL+	CL+	CL+
Variance(3) opt2.	X,CL-	CL-	CL-	CL	CL	CL
Variance(5) opt2.	X,CL	CL	CL	CL+	CL+	CL+
Local Statistic	CL-	CL	CL	CL	CL+	CL+
Local Statistic opt2.	CL-	CL	CL	CL	CL+	CL+

For each method, the simulation results for different Signal to Noise Ratios are discussed, and the problems of applying a method to synthetical images are pointed out. The discussion on the result analysis does not refer to any specific figures and tables. It is up to the reader to follow the analysis in the above given graphs. A

statement of comparison is given after the discussion of the results of the real image experiments where advantages and disadvantages become more obvious.

In the discussion below, the judgement whether a method succeeds or fails is based on the following approximated criteria:

Pixel stability fails when PS > 0.8

Geometric stability fails when GS > 0.1

Repeatability fails when STD > 0.1

3.5.1 Threshold(a) method

Because of the fact that the edgepixels are found by contour following, the measure of pixel stability is consistent over the entire range of SNR tested. The step in the measure of geometric stability between the image of SNR=∞ and the images with noise is due to the fact that the threshold value is chosen to be half the intensity value between object and background intensity. This of course interacts with the model of the edge, where the pixel intensity of the middle of the edge corresponds with the chosen threshold. The increase of noise in the image will be detected immediately. This is a problem of simulation. For low SNR the edge of the circles becomes rougher. In the low contrast case for SNR=10, no closed contour is observed and therefore the contour following algorithm fails.

3.5.2 Threshold(b) method

By studying the quantitative performance measures, thresholding(b) performs well for the range SNR>150. But the qualitative measures show that for SNR<=2500 the edge contour is broken, but on the other hand,

the edge is smooth and close to the smoothness of an edge of a perfect circle. For SNR<=40, the failure of this method is observed due to occurrence of noise pixels.

3.5.3 Roberts Edge Filter method

The ideal image of the contour without noise consists of a triple edge. A closed single contour edge cannot be obtained due to the characteristics of Roberts edge filter (see also [9]). For 625 < SNR < 10000, less pixels are used to build the edge which results in pixel stability > 1. Pixel and geometric stability shows failure for SNR<=156, in the low contrast case. In the high contrast case, the measure of pixel stability shows failure for SNR<=40, but the measure of geometric stability and repeatability reveal that this method fails for even SNR<=156. In this case pixels which do not belong to the edge exceed the gradient threshold, but not significantly enough for the pixel stability measure to show failure. This is one example where it is demonstrated, that the measure of pixel stability does not imply geometric stability.

3.5.4 Variance(n) method

Compared to variance(3), variance(5) detects a smoother edge. Variance(3) performs well for 156 < SNR < ∞ for both options. By using variance(5) option 1 extends the range of use to 40 < SNR < ∞. Option 2 instead, only performs as well as variance(3). It can be seen that for low SNR, better results are obtained by specifying the intensity range , where the pixels are processed (option 1) combined with a larger neighborhood size of pixels to smooth the noise pixels. The larger the

n x n variance window, the more features can be enhanced properly from noisy images. For option 2 and SNR = 10 in the low contrast case it is demonstrated, that geometric stability does not imply pixel stability. In this case the specification of the approximate knowledge of the edge influences the edge pixel enhancement. Every pixel in the specified area is detected as an edge pixel, therefore this method seems to be geometrically stable for SNR=10.

3.5.5 Local Statistic method

Lee's Local method differs from thresholding(a) in that a preprocessing of the image is done. The result of the simulation show that this method performs well over the entire range. It is the best method with respect to geometric stability. The computation time to process an image is higher compared to variance(3) and variance(5). For SNR<=40 the detected edge is rough, but still the radius estimation algorithm gives repeatable results. Within the tested range there is no significant difference between option 1 and 2 observed.

3.6 REAL IMAGE EXPERIMENTS

The different performances of the methods, become more obvious by applying them to real world images with low Signal to Noise ratio acquired by the CCD camera. For this purpose images of a brass cylinder with typical metal texture surface and a wood block with texture lines on a white background were processed. The input parameters for the methods were found by trying to obtain a closed contour without occurence of noise pixels. For every method the qualitative results in means of images are shown in Figures 28-41. The pictures are

photographs taken from a black and white display monitor. Unfortunately, the quality of these pictures is not very high. The stripes are produced by the display monitor and are not part of the images.

3.6.1 Circular Objects

The original gray scale image of the brass cylinder with textured surface is shown in Figure 28. The scene is illuminated by diffuse overhead lighting. The Signal to Noise ratio of this image is determined to SNR=156. In the lower part of the image the threshold(a) method (Figure 29) fails. The circle is flattened. This occurs due to the shadow zone and the low contrast, which can be seen in the original gray scale image. By comparing the contour image extracted by the local statistic method (Figure 30), with the result of the threshold(a) method, the advantage of the local statistic method is observed. The shape of the circle is more acceptable than that detected by the threshold(a) method. Still in the lower left corner, local statistic fails, and noise is introduced in the contour. The contrast of the texture of the cylinder is higher than the contrast of the edge in the lower left and right part of the image. Therefore the variance threshold for the variance(5) method (Figure 31) can either be chosen to reject noise pixels and fail to detect a close contour or vice versa. The same problem occurs using the Roberts edge filter (Figure 32). Compared to variance(5), the algorithm is more sensitive to contrast variations and therefore more noise pixels are detected. The threshold(b) method fails as well (Figure 33). The intensity range of the edge pixels cannot be specified accurate enough to detect the edge

properly. Noise pixels occur in the background as well as in the object.

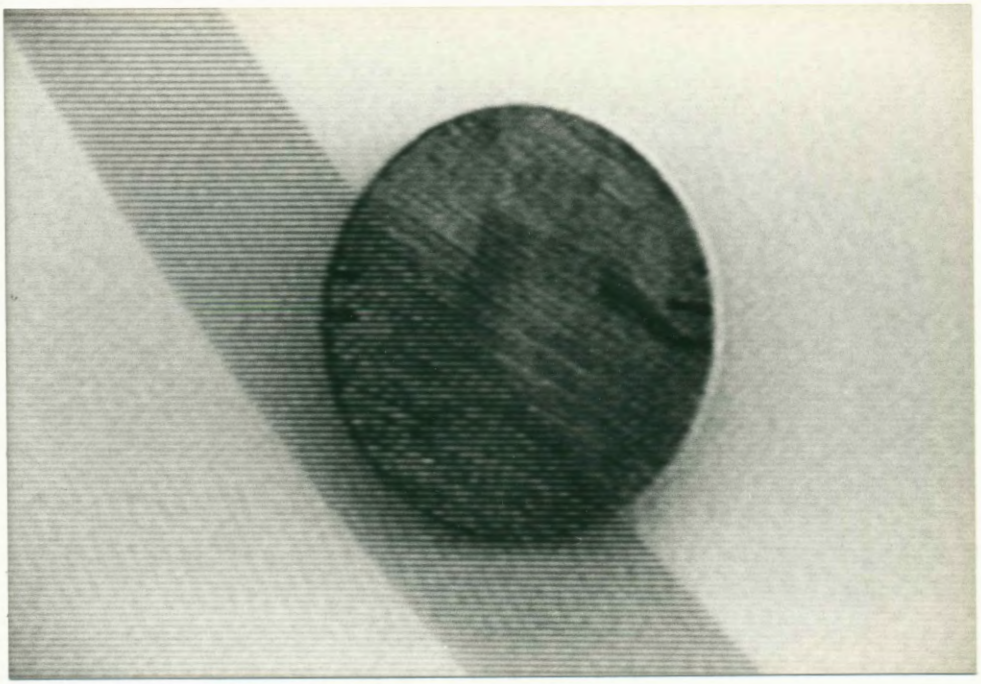


Figure 28. Gray scale of image of brass cylinder with textured surface

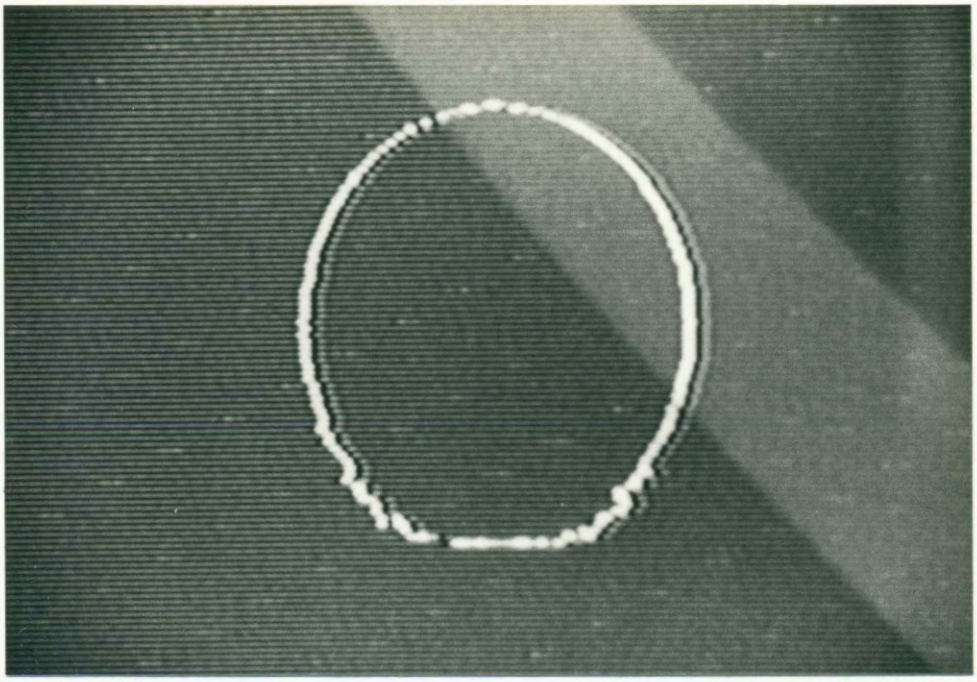


Figure 29. Contour image extracted by the threshold(a) method

Figure 30. Contour image extracted by the local statistic method

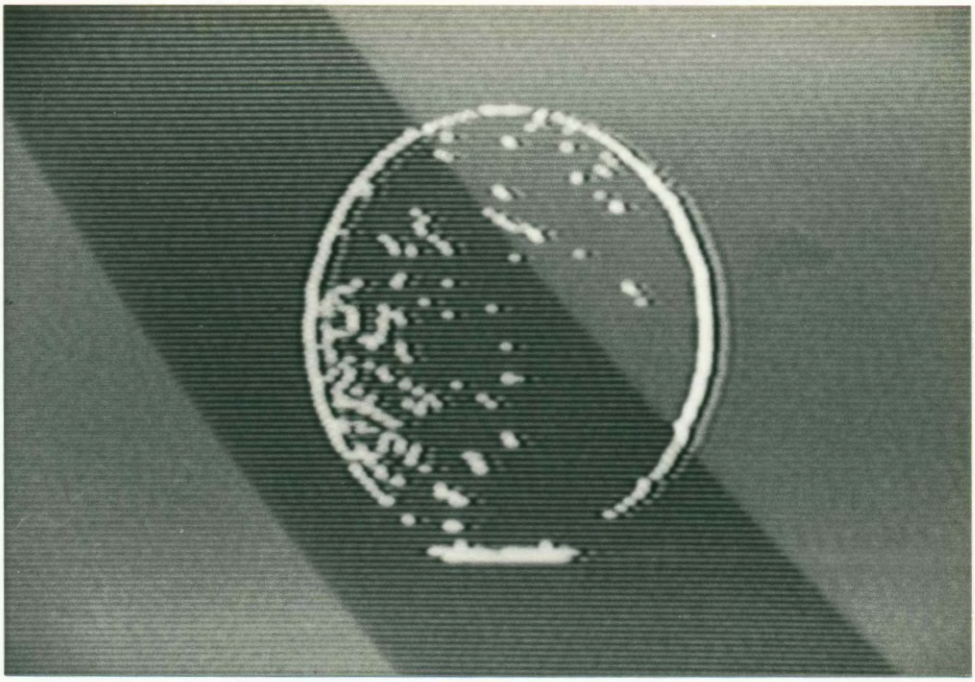


Figure 31. Contour image extracted by the variance(5) method

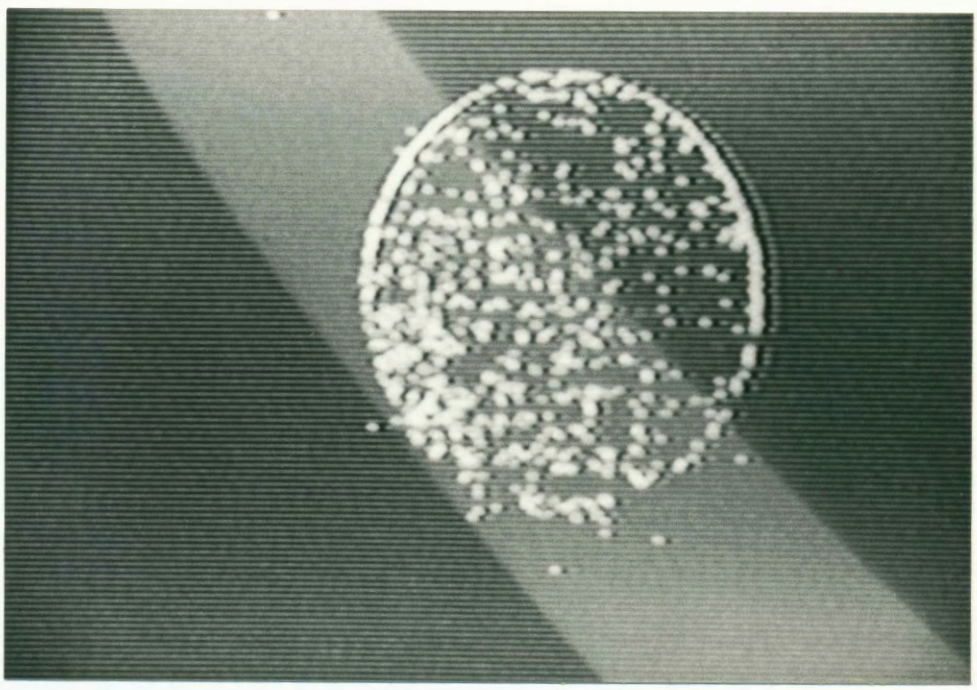


Figure 32. Contour image of edge detected by Roberts edge filter

Figure 33. Contour image extracted by the threshold(b) method

3.6.2 Rectangular Objects

The original gray scale image of the wood block is shown in Figure 34. The Signal to Noise ratio is SNR=156. Different than in the case of circular edges, the geometry of a straight line is related to the geometry of a CCD element. A straight line edge occurs perfectly on a binary contour image if the edge line is exactly parallel to the rows and columns of the CCD elements. Otherwise, a straight line contour with steps is detected and no perfect contour image presentation can be obtained. The threshold(a) method (Figure 35) detects a closed contour. The upper horizontal edge is rough. Since pixels of the textured surface have the same intensity as the background, no threshold can be found which completely divides the set of object pixels from the set of background pixels. The upper horizontal edge is detected as a jagged

line. By preprocessing the gray scale image with the local statistic filter these problems disappear. A sharp closed straight line contour is detected (Figure 36). Figures 37, 38, and 39 show the result of the variance(n) method using 3 x 3, 5 x 5, 7 x 7, neighboring pixels. With increasing n the occurrence of noise pixels is reduced. For the same reason as in the case of detecting a circle, Roberts edge filter (Figure 40) and the threshold(b) methods (Figure 41) fail.

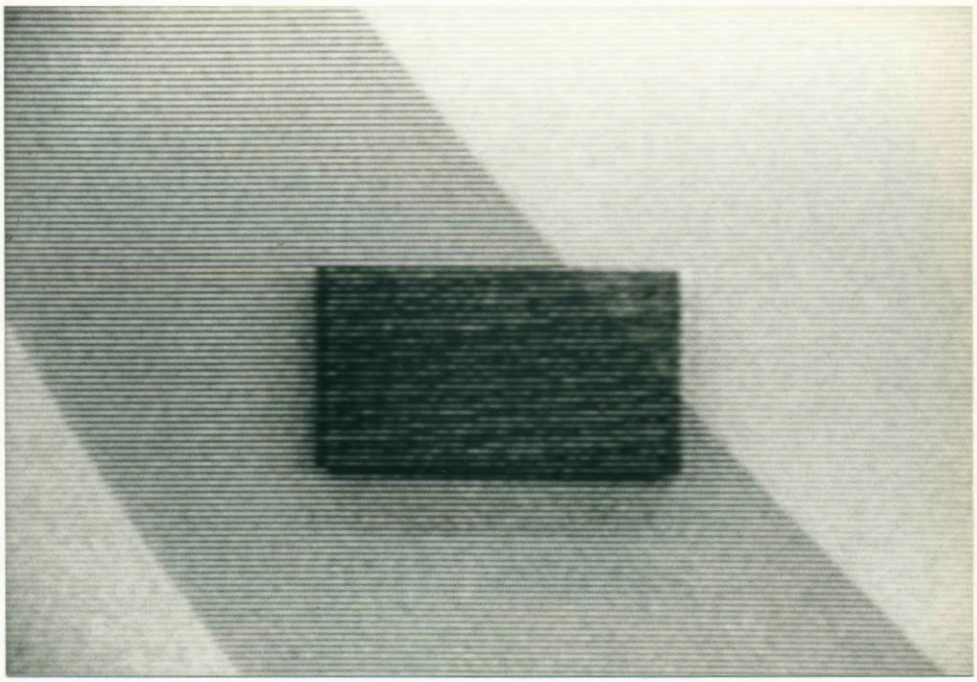


Figure 34. Gray scale image of rectangular shaped block of wood

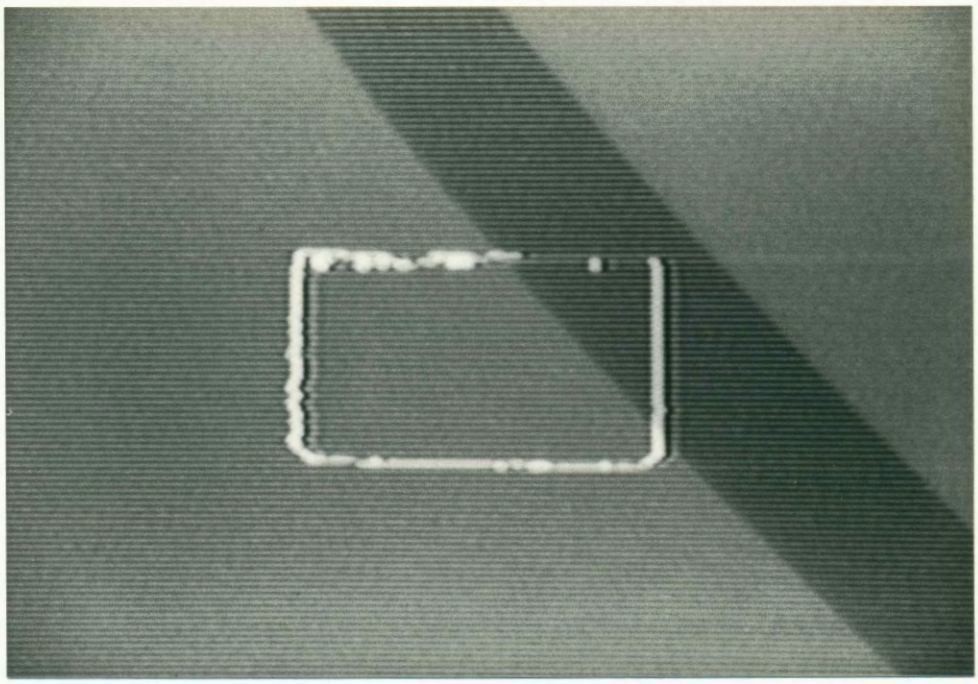


Figure 35. Contour image extracted by the threshold(a) method

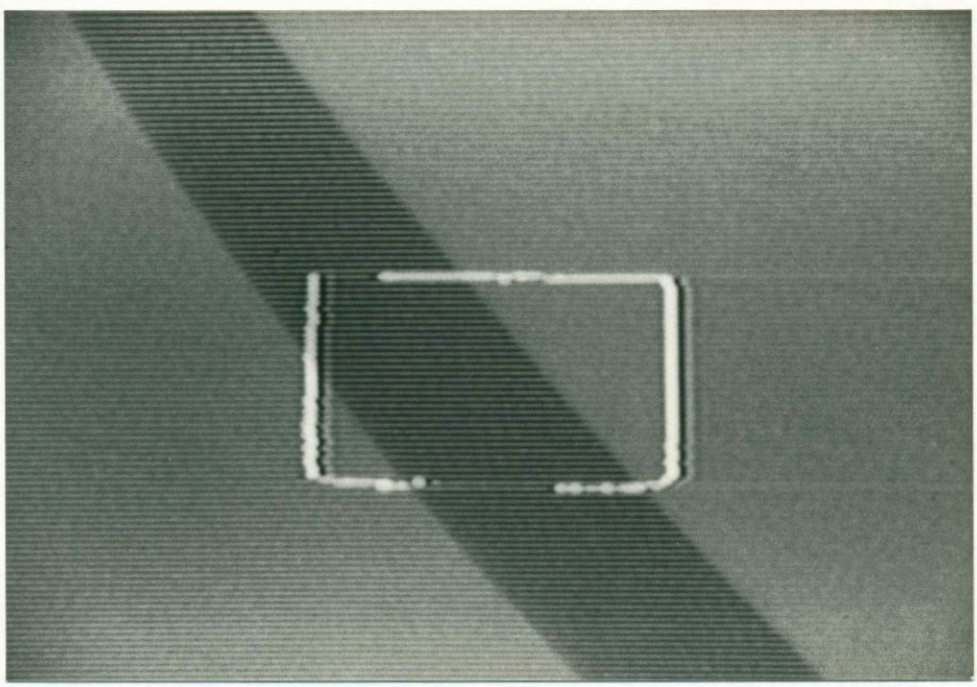


Figure 36. Contour image extracted by the local statistic method

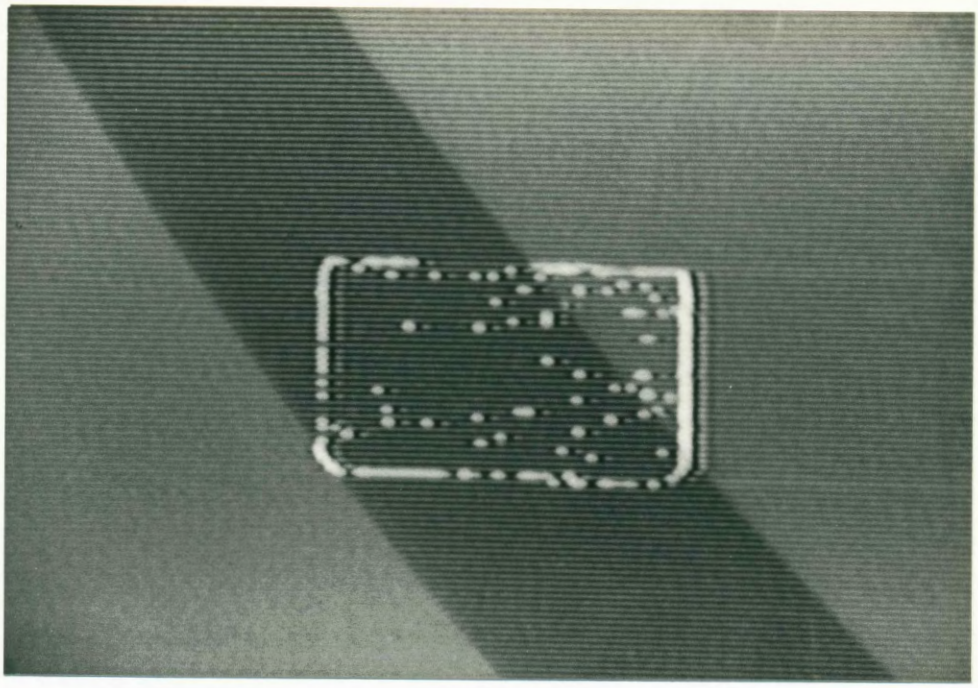


Figure 37. Contour image extracted by the variance(3) method

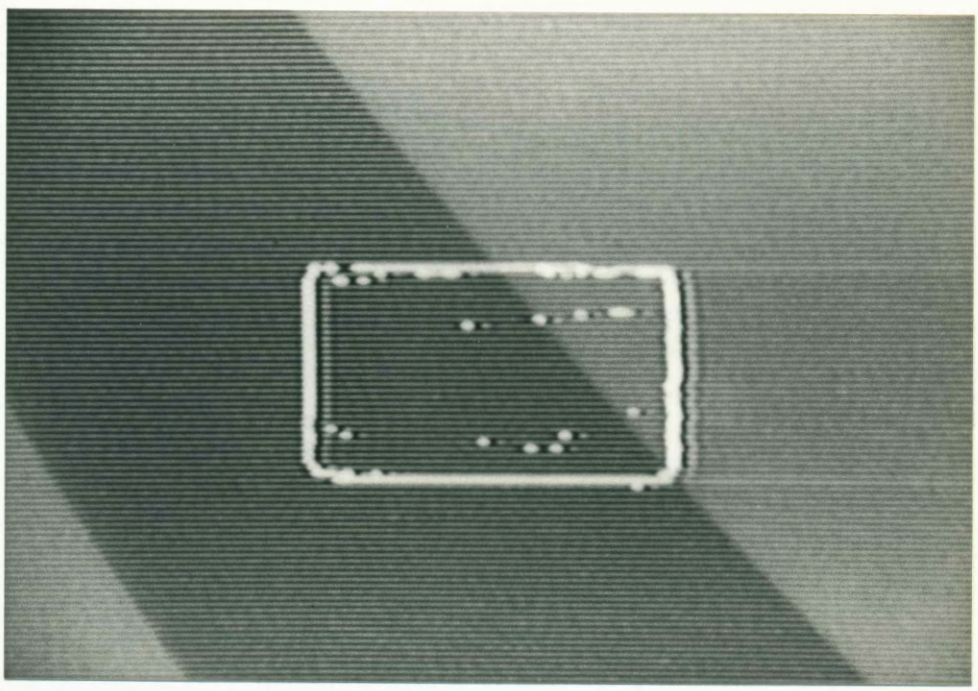


Figure 38. Contour image extracted by the variance(5) method

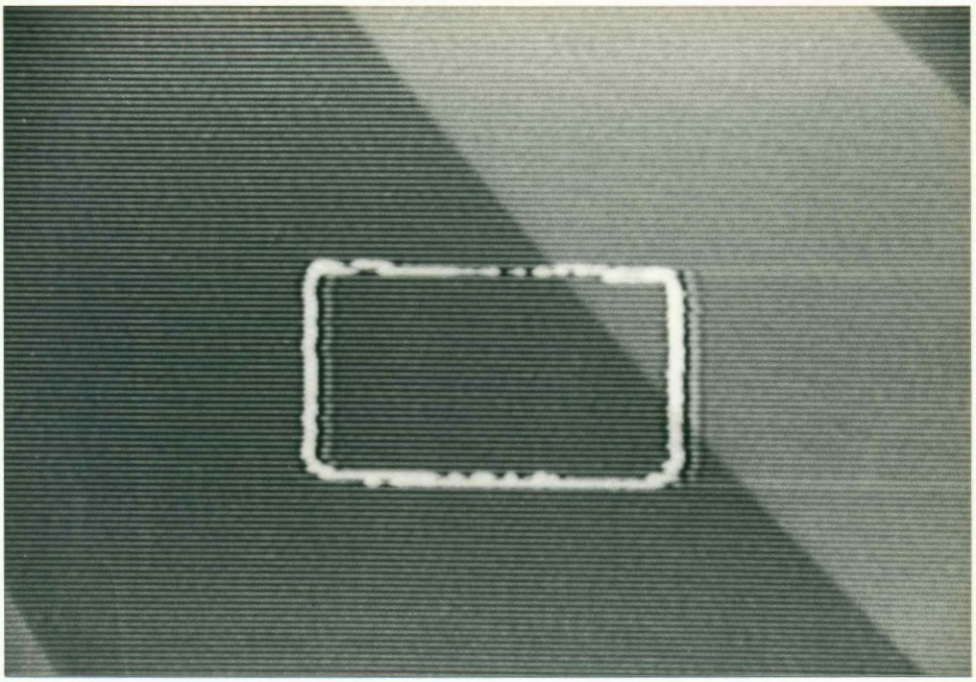


Figure 39. Contour image extracted by the variance(7) method

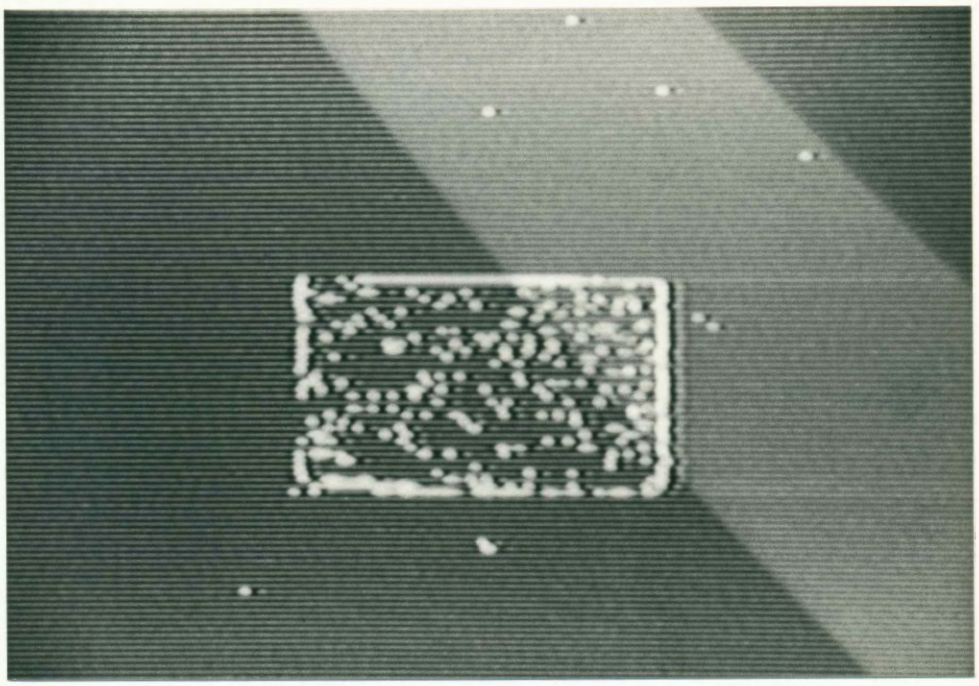


Figure 40. Contour image of edge detected by Roberts edge filter

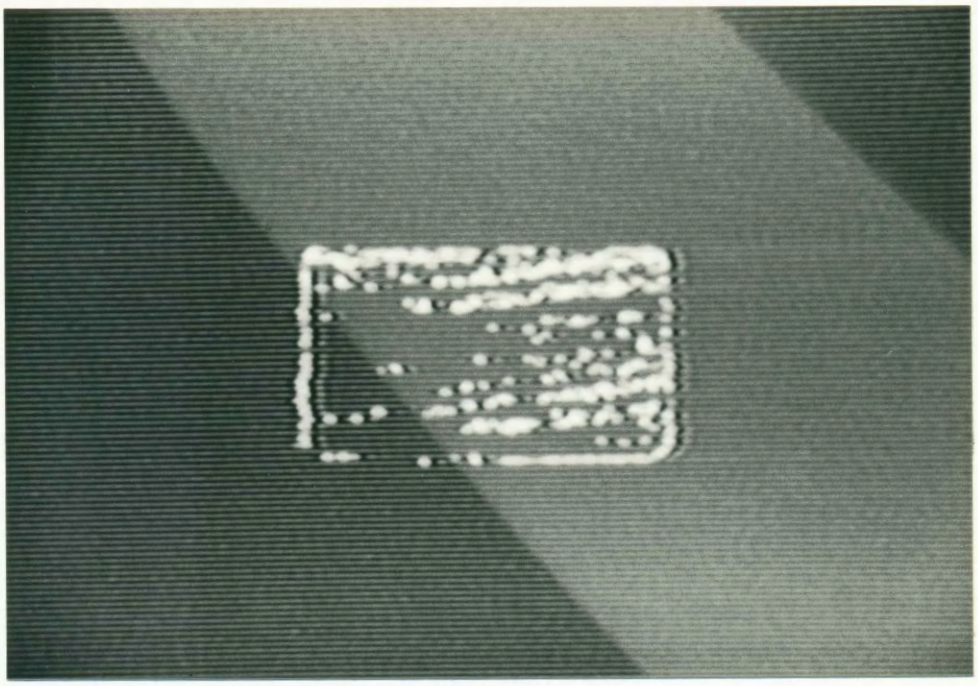


Figure 41. Contour image extracted by the threshold(b) method

3.7 STATEMENT OF COMPARISON

The tested algorithms can basically be divided into 3 groups.

- threshold(a), local statistic filter
- variance(n), Roberts edge filter
- threshold(b)

Threshold(a) and the local statistic method both result in a binary image, from which the edge contour is extracted. For increasing noise level no exact threshold can be found. By preprocessing the image with the local statistic filter, noise pixels are smoothed by averaging, using a subset of neighborhood pixels, which represent the most likely side of the edge where the pixel to be filterd belongs to. Therefore the local statistic filter performs far better for images with low SNR

and textured edges than thresholding(a).

The variance(n) method uses, like the local statistic method, larger areas of neighborhood pixels to make a decision about an edge but without preprocessing the image. The larger the window size, n, the more capable is the method for detecting edges in images with low SNR. Another advantage of increasing n is the smoother representation of the edge. Since noise pixels are rejected, the variance threshold can be optimized with regard to a sharp representation of the edge and a closed contour. Edges of textured objects can be detected for large n, as well, but still the local statistic method is preferable for this task, due to the use of gradient direction information. Processing the next larger size of neighborhood increases the computation time by a factor of $(n/n-1)^2$. The variance window size n should not be too large, otherwise sharp edges cannot be detected detailed enough. The change of the gradient of the edge line within the window should be small. Roberts filter is similar to the variance(n) method, but uses less pixels to decide whether or not a point belongs to an edge or not. Based on 2 cross gradients of 4 pixels the new pixel value is calculated. These are less pixels than the variance(3) methods uses to establish a measure of edge strength. Therefore Roberts Edge filter is more sensitive to noise and texture. Roberts edge filter should be used for images with high signal to noise ratio. Sharp edges are detected more accuratly with this method than with filters using large window sizes.

Threshold(b) is a weak method for the purpose of dimensional inspection if the image is noisy or shows textured parts. This method

can only be used for accurate measurements for images with high SNR. However the computation time for the detection process is minimum because the operation is performed by hardware and no contour has to be processed as in the case of threshold(a). This method should be used for inspection cycles with high frequency. As threshold(a), threshold(b) relates only to the intensity value of one pixel without regard to any spatial relationships. Threshold(a) is a stronger method in detecting edges, because the contour following algorithm is only influenced by noise pixels occurring next to the edge. Threshold(b), Roberts edge filter, and the variance method detect noise pixels occurring in the area of measurement besides the extracted edge pixels. This results in weak geometric stability as soon as noise pixels are enhanced.

The methods were not compared with regard to the computational effort for the process of edge detection. The processing time changes with the type of image. First, criteria for standard images have to be developed to allow a successful benchmarking operation. To give the reader a general idea, a ranking of the methods is given in increasing order for the computation time: threshold(b), threshold(a), Roberts edge filter, variance(3), variance(5), local statistic filter.

CHAPTER IV

CALIBRATION

Besides choosing the right image analysis method for a certain type of application, the user of a vision system has to calibrate the camera setup. If the plane of the camera lens is not parallel to the circle surface, a perfect circle would appear as an ellipse on the image. This effect has to be taken into account and compensated for. Therefore, calibration is another important factor in determining the accuracy of a vision system.

Calibration is defined in The Manual of Photogrammetry [22] as:

> *The act or process of determining certain specific measurements in a camera or other instrument or device by comparison with a standard, for use in correcting or compensating errors for purposes of record.*

To obtain knowledge about the real dimensions of a part which has been measured, the image information about this part has to be changed into a real world representation. This is done by transforming pixel information from the image coordinate system to the real world coordinate system. To be able to do this requires knowledge about the transformation parameters. The process of finding these parameters is called calibration.

To calibrate a vision system, the following equipment information must be acquired:

1) highly accurate calibration pattern coordinates (real world knowledge)
2) representation of these coordinates in the image plane
3) a model to calculate the calibration parameters.

4.1 A MODEL

The calibration process is done in four steps (Figure 42). The world coordinates of a highly accurate pattern are transformed into the camera coordinate system (step 1), and then into the coordinate system of the image plane (step 2). Compensating for distortion transforms the points into the form where the CCD elements of the camera produce the signal (step 3). Then, the coordinates are transformed into computer coordinates (step 4), which are displayed on the monitor of the vision system.

The coordinate system setup is shown in Figure 43. All coordinate systems are right-handed. The z-axis of the world coordinate system points in the opposite direction of the z-axis of the camera coordinate system. The x,y planes of the camera and the image coordinate systems are parallel. Their origins have the same x,y coordinates and the distance between the planes is f, the focal length.

4.1.1 Transformation from world to camera coordinate system

To transform coordinates from the world to the camera coordinate system, the points have to be translated by

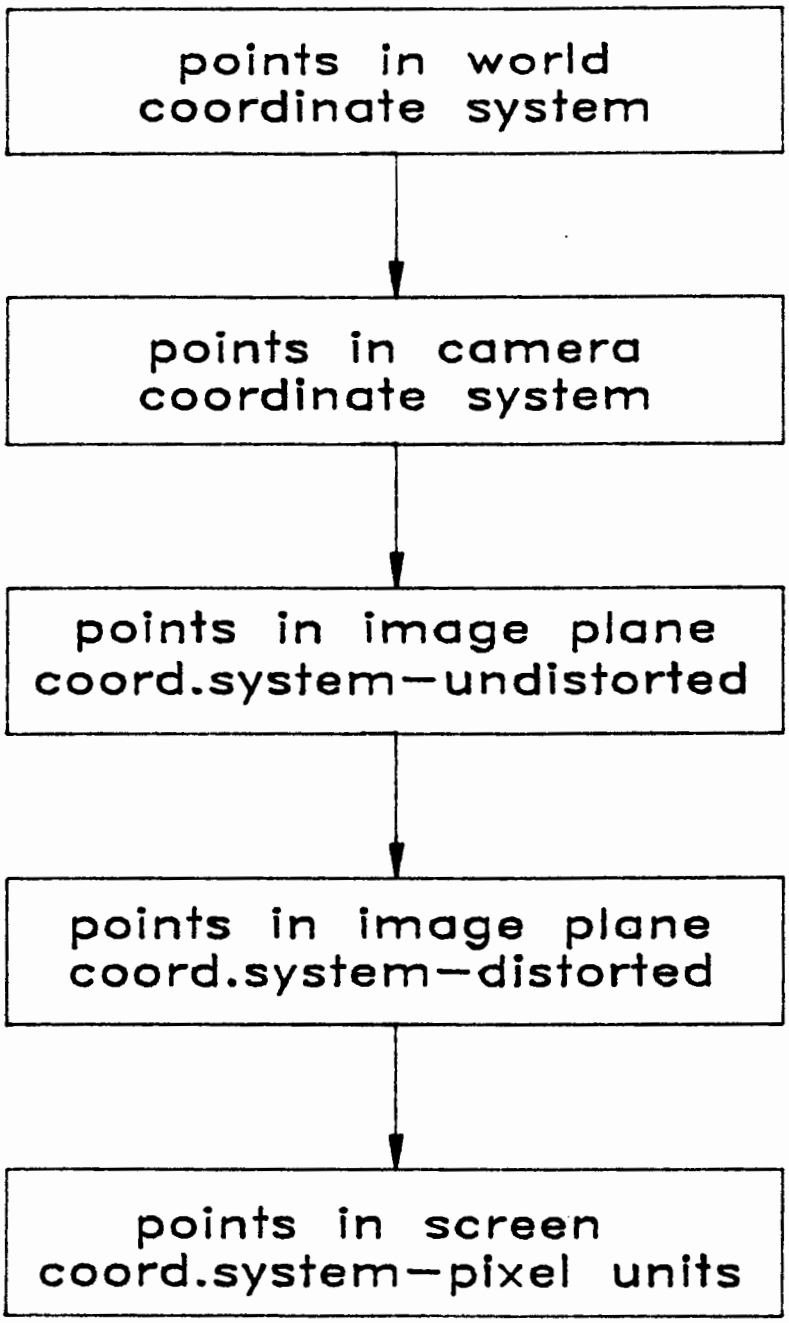


Figure 42. Calibration in 4 steps.

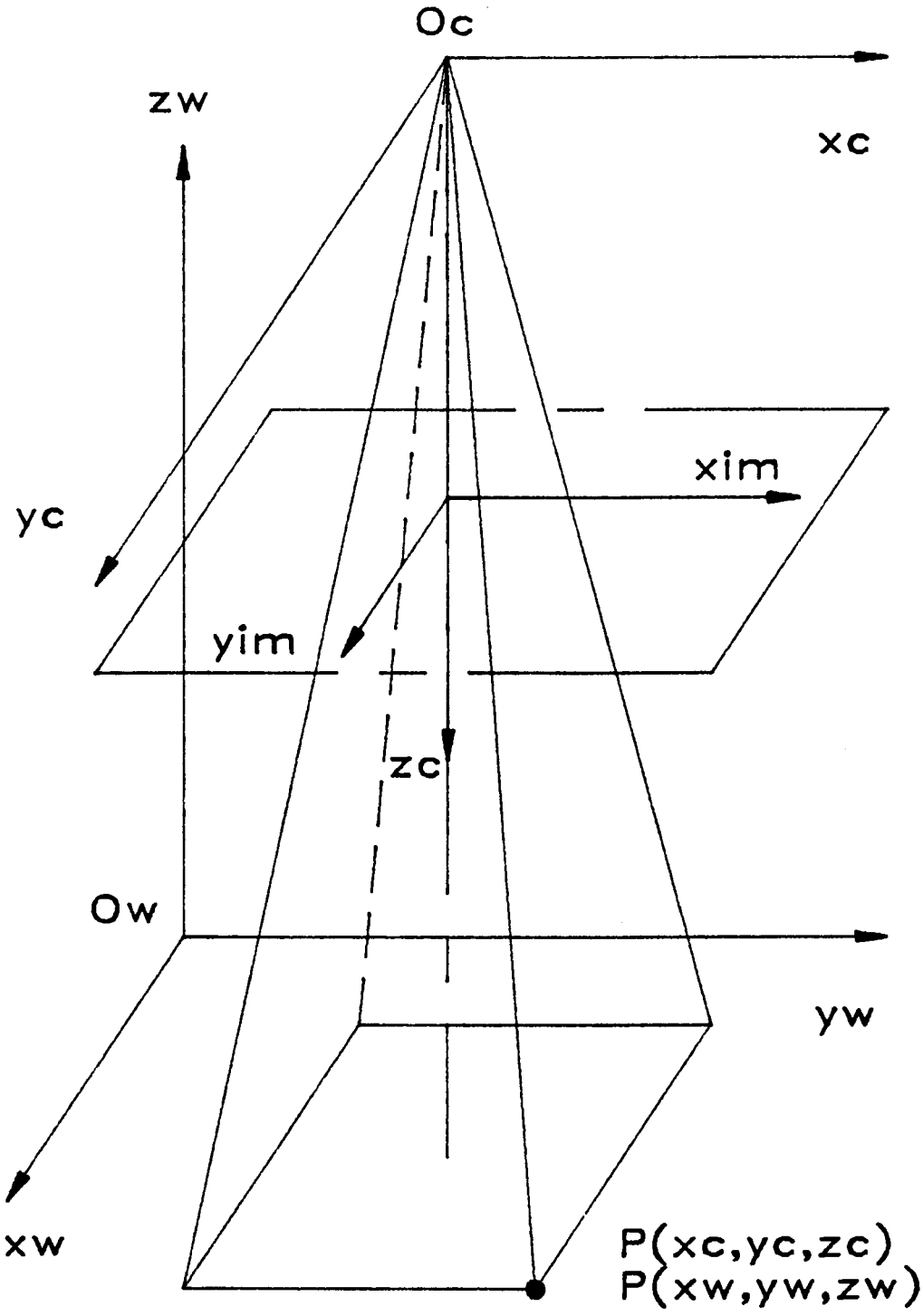


Figure 43. Orientation of coordinate system for calibration.

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

where T_x, T_y, and T_z are the coordinates of the origin of the world coordinate system in the camera coordinate system representing the translation vectors in x,y,z direction.

The translated points are rotated by using the rotation matrix

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}$$

where rl - r_9 are the components of the rotation matrix, as a function of the rotation angle θ, ϕ, ψ of the x,y,z axis.

This results in the composite transformation matrix,

$$T_{wc} = \begin{bmatrix} r_1 & r_2 & r_3 & T_x \\ r_4 & r_5 & r_6 & T_y \\ r_7 & r_8 & r_9 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation equation results in

$P_c = T_{wc} \cdot P_w$, where

P_c = point in camera coordinate system, and

P_w = point in world coordinate system.

The transformation matrix in the form of rotation angles yields

$P_c = R_z(\psi) \cdot R_y(\phi) \cdot R_x(\theta) \cdot T_R \cdot P_w$, where

θ, ϕ, ψ are the Euler angles, known as yaw, pitch, and roll, and

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The composite rotation matrix $R = R_z(\psi) \cdot R_y(\phi) \cdot R_x(\theta)$ results in:

$$R = \begin{bmatrix} \cos\psi\cos\phi & -\sin\psi\cos\theta+\cos\psi\sin\phi\sin\theta & \sin\psi\sin\theta+\cos\psi\sin\phi\cos\theta & 0 \\ \sin\psi\cos\phi & \cos\psi\cos\theta+\sin\psi\sin\phi\sin\theta & -\cos\psi\sin\theta+\sin\psi\sin\phi\cos\theta & 0 \\ -\sin\phi & \cos\phi\sin\theta & \cos\phi\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the coordinate system setup in this project, the Euler angles are approximately:

x : $\theta \approx 180^0$

y : $\phi \approx 0^0$

z : $\psi \approx 90^0$

and yield the rotation matrix

$$R \approx \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

4.1.2 Transformation from camera to image plane coordinate system for undistorting optical systems.

Using the model of the pinhole camera geometry, in other words, the fact that the x,y plane of the camera coordinate system is parallel to the image plane but translated by $|f|$, the focal length, the following relationship can be established:

$$\frac{X_{im}}{X_c} = \frac{|f|}{|T_z|}$$

where T_z is the translation component of the camera coordinate system in z-direction with respect to the origin of the world coordinate system. X_{im}, Y_{im} are the coordinates of a point in the image coordinate system, obtained by intersecting the x,y plane of the image coordinate system with a ray, starting from a point, with the coordinates x_c, y_c expressed in the camera coordinate system.

This results in:

$$X_{im} = X_c \frac{|f|}{|T_z|}$$

and similarly in:

$$Y_{im} = Y_c \frac{|f|}{|T_z|}$$

4.1.3 Transformation of points from undistorted to distorted image coordinates.

Two kinds of lens distortion can be observed, radial and tangential. The radial lens distortion can be further divided in barrel distortion and pin cushion distortion. For the fundamental theory of distortion and the derivation of equations, the Manual of Photogrammetry[22] is recommended. The radial lens distortion is modeled as an odd-powered polynomial [22],

$$\Delta r = k_1 \cdot r^3 + k_2 \cdot r^5 + k_3 \cdot r^7 + \ldots$$

using the relationship

$$x' = x - x \frac{\Delta r}{r}$$

$$y' = y - y \frac{\Delta r}{r}$$

where x',y' are the corrected image coordinates of the undistorted image. Tsai [23] proposes that only K, the radial lens distortion coefficient, should be considered and that tangential distortion can be neglected.

4.1.4 Final Transformation into Computer Coordinate System.

To obtain the computer coordinates, the calibration points have to be transformed from distance units to pixel units. Figure 44 shows the process of scaling and transformation into computer coordinates. The image of the CCD element is digitized and scaled to fit the frame buffer. Since the origin of the image plane is placed in the center of the computer coordinate system, C_x, C_y, the scaled points in computer

coordinate system are translated by C_x, C_y.

The scale factor to transform distance to pixel units is

$$\frac{\text{distance in x of computer coordinates}}{\text{distance in x in undistorted image coordinate system}} = \frac{1}{d_x^*}$$

setting the scale factor in a similar way for y, yields the transformation equations

$$x_f = \frac{1}{d_x^*} \cdot x' + c_x \qquad (*)$$

$$y_f = \frac{1}{d_y^*} \cdot y' + c_y$$

where x_f, y_f are the computer image coordinates.

Since the number of pixels of the CCD element N_{cx} are mapped onto the frame buffer with the computer sampling N_{fx},

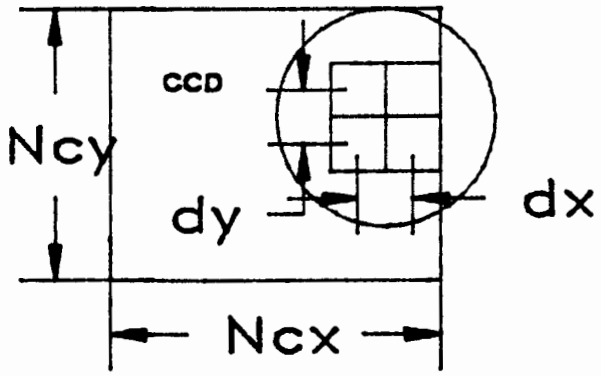
$$d_x^* = d_x \frac{N_{cx}}{N_{fx}}$$

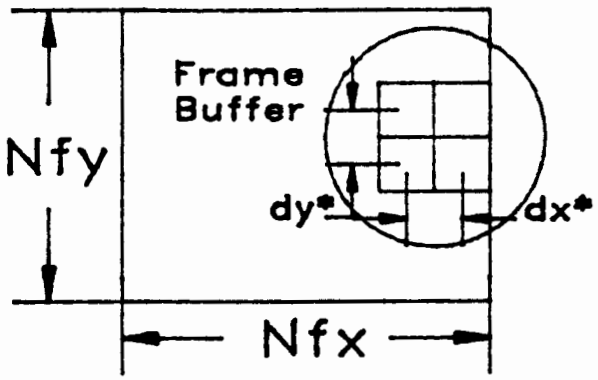
where d_x is the distance of two adjacent CCD elements in x direction.

Whereas in y direction, every row is processed by vertical scanning and therefore,

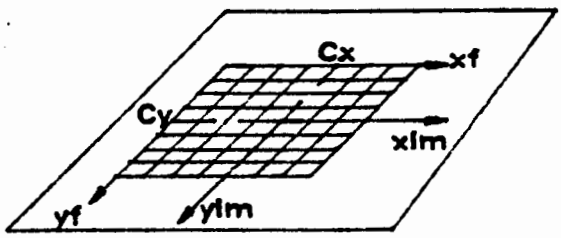
$$d_y^* = d_y$$

and d_y is the distance of two adjacent CCD elements in y direction.





$$dx^* = dx(Ncx/Nfx)$$
$$dy^* = dy$$



$$xf = x'/dx^* + Cx$$
$$yf = y'/dy^* + Cy$$

Figure 44. Scaling and transformation into computer coordinate system.

For the CCD camera used in this project, N_{cx}=512, N_{cy}=492, d_x=17μ and d_y=13μ. The signal of the 512 elements in x direction of the CCD element is digitized on 640 pixels. From the 492 element lines in y direction of the CCD element, only N_{fy}=480 are used, and the signal produced by the rest is not used to form the image.

4.2 COMPUTING THE CALIBRATION PARAMETERS

4.2.1 Nonlinear Optimization

In 4.1 (A Model), formulas relating the world coordinate to the computer coordinate system are derived. The eight unknown parameters are found by numerical calculations. The numerical solution to the calibration problem can be found by the use of nonlinear optimization of the calibration parameters θ, ϕ, ψ, T_x, T_y, T_z, k_1, f. The calibration points in the world coordinate system are transformed in four steps into the computer coordinate system using the determined equations. At this point, a good initial guess is required. The squared sum of the difference between the observed pixels and the calculated values represented in the computer coordinate system is minimized by optimizing the calibration parameters.

The disadvantage of using nonlinear optimization is that the computation time to optimize the variables is too long for practical purposes. In addition, by using an initial guess, the minimization function will most likely diverge.

4.2.2 Two-stage solution by Tsai [23]

Tsai developed a useful method to compute the calibration parameters. This method was studied and implemented during this

project. Tsai's model differs from the model derived above, in that in equation (*), an uncertainty factor S_x is introduced to compensate for timing mismatch between image acquisition and camera scanning hardware. For the equipment used, S_x=1.042. A short outline of how this two stage calibration is implemented is described below.

The calibration takes into account the parameters of the rotation matrix r_1 to r_9, the component of the translation matrix T_x, T_y, T_z, the focal length f and the radial distortion coefficient K_1.

By introducing the "Radial Alignment Constraint," a solution for

$$\frac{r_1}{T_y}, \quad \frac{r_2}{T_y}, \quad \frac{r_4}{T_y}, \quad \frac{r_5}{T_y} \quad \text{and} \quad \frac{T_x}{T_y}$$

can be obtained by solving an overdetermined system of linear equations.

Radial alignment constraint

Because of the fact that the x,y-plane of the camera coordinate system and the x,y-plane of the image coordinate system of the model are parallel, the product of the two vectors, representing a point in the world and the image coordinate system, is $(x_{im}, y_{im}) \times (x_c, y_c) = 0$. This results in an overdetermined set of linear equations. To solve this overdetermined system, the method of singular value decomposition [24] has been used in this project. This method gives very accurate results and has the advantage of obtaining the closest solution in the case of an ill-conditioned matrix.

Using the fact that an orthonormal 3x3 matrix has a unique 2x2 submatrix, differing by a scale factor no other than ±1, the absolute value of T_y is calculated. The rest of the components of the rotation

matrix r_3, r_6, r_7, r_8, r_9 are solved using the orthonormal property. Since the focal length is $f>0$ and $T_z>0$, the coordinates in world and image plane must have the same sign. If this is not the case, the sign of T_y must be changed. A second overdetermined system of equations is solved for f and T_z. The calibration plane and image plane must not be exactly parallel since the set of equations would then become linearly dependent. If the focal length is $f<0$, then the signs of r_3, r_6, r_7, r_8 have to be reversed and f and T_z are calculated again. By using optimization, f, T_z, and k_1 are determined. For this purpose, the conjugate gradient method of Polak-Ribiere has been implemented [24].

The source code of the implemented calibration algorithm is included in Appendix F. Because a highly accurate calibration pattern was not available at the time of testing time, the algorithm was tested on pseudo pattern with 12 calibration points. The algorithm was verified with regard to this accuracy. The computation time for the numerical solution of the calibration on a PC-AT 12.5 MHZ was approximately 2 seconds.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

Different image analysis methods have been studied and compared. But how do the results of the comparison of the methods and the calibration related to the overall performance of a vision system for dimensional inspection? Figure 45 shows three main areas, which determine the accuracy of the measurement. These three areas are:

- Image analysis methods used for measurements
- Calibration and its accuracy
- Resolution of the system

Image Analysis Methods

The main emphasis in this research project was on the comparison of different image analysis methods. A decision model to select the right method for a certain application has been proposed. The experimental results show the range where the tested methods perform well, and the range where they fail. The quantitative real world experiments illustrate the conclusions drawn out of the simulation experiments. With the result of the simulation in Chapter III, it is possible to choose the right method for an application or specific image scene. For accurate dimensional inspection, the contrast, i.e. the difference between the part and the background intensity should be as large as possible. High contrast inspection scenes can be achieved by

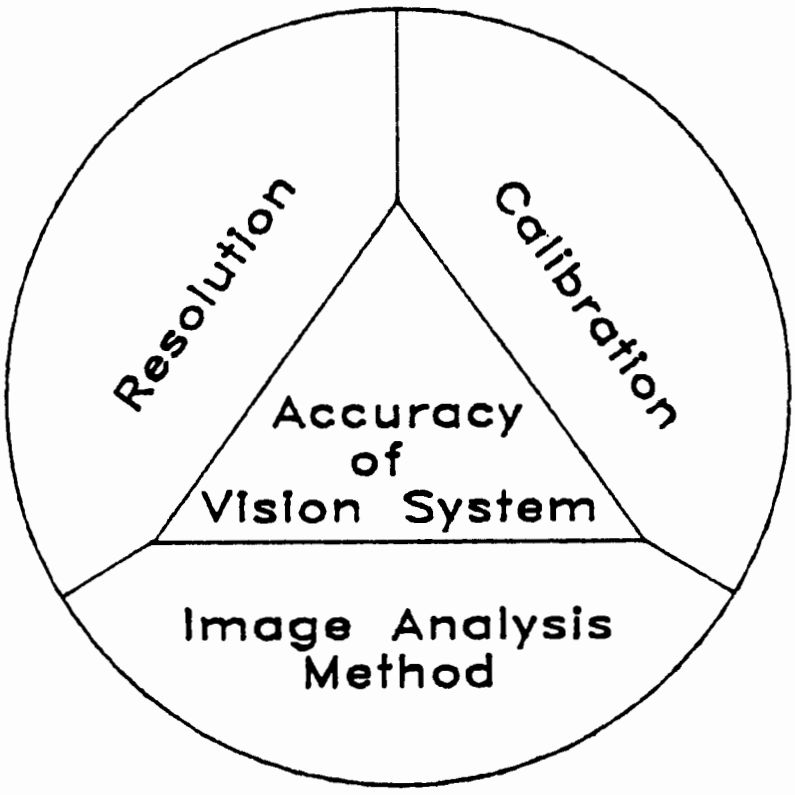


Figure 45. Accuracy of vision systems for dimensional inspection

using a diffuse light source mounted below the part. In applications where this type of lighting cannot be established, diffuse overhead light should illuminate the scene as uniformly as possible. In general it can be said, that the lower the Signal to Noise ratio of a scene, the more difficult is it to detect the features of the image, and the higher the computational effort. Every image analysis method introduces errors in terms of the qualitative measures such as pixel stability, geometric stability and repeatability of the outcome of the measurement.

Calibration

Image coordinates, in units of pixels have to be transformed to world coordinates in units of dimensional measures, to obtain "Real

World Knowledge" of the detected edge of the part. For this purpose a calibration algorithm has been implemented. Its computational speed and the general setup of the calibration model suggest its use in the field of robotic vision system, where the calibration parameters have to be calculated with every movement of the robot. As the image analysis method, the process of calibration contributes errors to the overall accuracy of the system. With increasing number of calibration points the error decreases [23].

Resolution of the system

The resolution of the system is expressed in terms of dimension per pixel and is the major factor in determining the accuracy of the dimensional inspection process with vision systems. Whether the measurement of a circle incorporates sub-pixel accuracy should be determined by testing the final accuracy of the measurement by taking into account the three main influence factors (Figure 45). In addition, the errors of the transformation of a circle from continuous to discrete space must be investigated.

One final experiment shows a trend towards the use of subpixel accuracy measures. A binary image of circles with different sizes were generated and the diameter was detected by thresholding(a). Figure 46 shows the ratio of the generated difference to detected difference versus the generated difference for different circle sizes in units of pixels. The approximate curve of the experimental points shows a sufficient response of the measurement system down to 0.5 pixel differences of the diameter. Gray scale systems should be even more sensitive due to the use of multiple intensity levels.

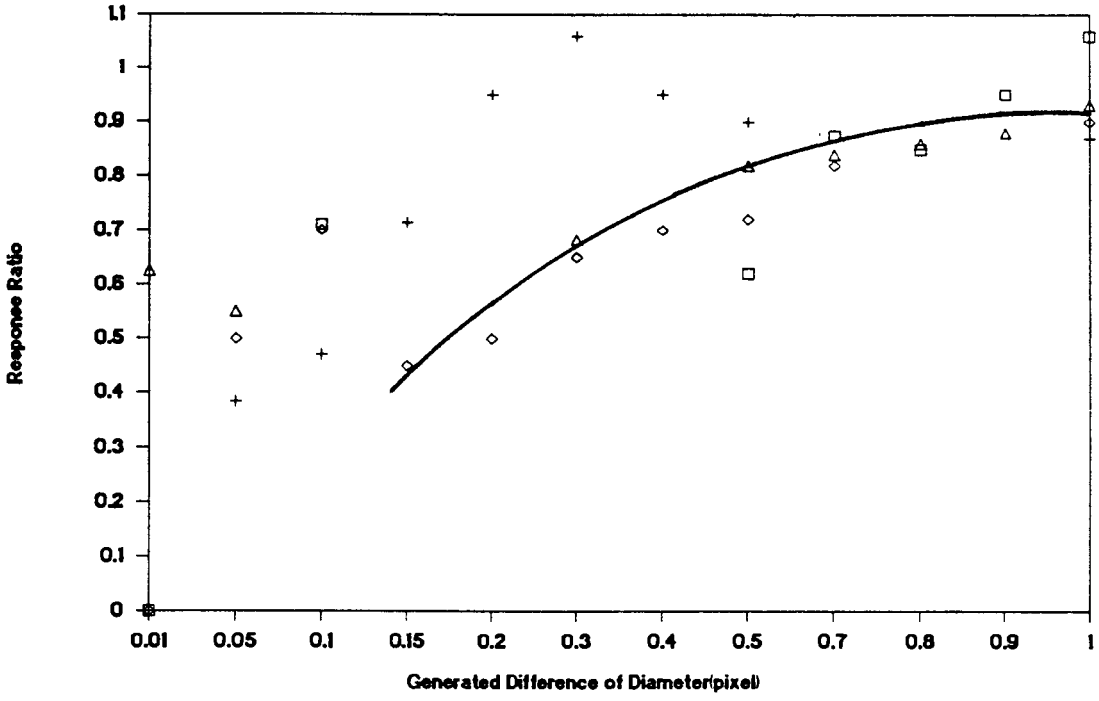


Figure 46.Resolution of circle in binary space for different diameters

To achieve knowledge about the accuracy of a vision system for dimensional inspection of circles, investigations have to be done in the resolution of the system and the use of subpixel accuracy. Then, the entire system must be tested, and an error model, including image analysis, calibration and resolution errors needs to be setup.

5.2 RECOMMENDATIONS

A system extension for measurements of objects with straight lines is appropriate. The detection process of straight line edges is similar to circular edges, but other approaches have to be found to obtain the geometrical parameters like distances between two edges. An interesting algorithm for the detection of straight line edges has been proposed by

Petkovic, et al [12].

Another important issue in the area of vision systems and image processing is the development of standards to compare hardware and software. Edge detection algorithms have to be compared using the same standard images. This would make it easier to choose the right method for a certain application. The Abingdon Cross Benchmark test [25] for comparison of image processing hardware is one step towards the standardization of comparisons in the field of vision systems.

The accuracy of vision systems for inspection can also be increased by using higher magnification lenses combined with precision tables, or by the use of subpixel measures. Much significant improvements can be realized by using high resolution CCD elements. In April 1989 the first CCD chip with 2048 x 2048 elements was introduced [26]. However to make this technology applicable for the task of dimensional inspection, more powerful computers have to be built to process the image in a reasonable amount of time.

REFERENCES

[1] F. Etesami, "Position Tolerance Verification Using Simulated Gaging", (to be published).

[2] J. Hollingum, "Machine Vision - The Eyes of Automation", Springer Verlag, 1984.

[3] T.Y. Young, K.S. Fu, "Handbook of Pattern Recognition and Image Processing", Academic Press, 1986.

[4] H.-J. Warnecke and C.P. Keferstein, "Optoelectronic sensor system based on picture processing", Technische Messen tm, 52nd Jahrgang, 9/1985.

[5] J.F. Bremner, "Automatic Vision Measurement System for the Inspection of Shapes Cut in Sheet Material", Second International Conference on Image Processing and its Applications, 1986.

[6] T.S. Huang, "Picture Processing and Digital Filtering", Springer Verlag, Berlin and New York, 1975.

[7] R.C. Gonzalez and P.A. Wintz, "Digital Image Processing", Addison-Wesley, 1977.

[8] A. Rosenfeld and A.C. Kak, "Digital Picture Processing", Academic Press, New York, 1982.

[9] T. Peli and D. Malah, "A Study of Edge Detection Algorithms", Computer Graphics and Image Processing 20, pp. 1-21, 1982.

[10] L.J. Van Vliet and I.T. Young, "A Nonlinear Laplace Operator as Edge Detector in Noisy Images", Computer Vision, Graphics, and Image Processing 45, pp. 167-195, 1989.

[11] M. Rueff and K. Melchior, "'Bildlib:' The Image Analysis Software at IPA," Journal of Robotic Systems, 2(2), pp. 179-198, 1985.

[12] D. Petkovic, W. Niblack, and M. Flicker, "Projection-based High Accuracy Measurement of Straight Line Edges", Machine Vision and Applications 1, pp. 183-199, 1988.

[13] "ITEX PCplus Programmer's Manual", Image Technology Incorporated, 1987.

[14] G.A. Baxes, "Digital Image Processing", Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[15] L.G. Roberts, "Machine perception of three dimensional solids", in Optical and Electro Optical Information Processing, pp. 159-197, M.I.T. Press, Cambridge, mass., 1965.

[16] J. Lee, "Digital Image Enhancement and Noise Filtering by Use of Local Statistics", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 2 March 1980.

[17] J. Lee, "Refined Filtering of Image Noise Using Local Statistics", Computer Graphics and Image Processing 15, pp. 380-389, 1981.

[18] G.S. Robinson, "Edge Detection by Compass Gradient Masks", Computer Graphics and Image Processing 6, pp. 492-501, 1977.

[19] G. Johannsen and J. Bille, "A Threshold Selection Method using Information Measure", Proc. International Conference on Pattern Recognition, Munich, 1982. IEEE Comp. Soc. Press, Silver Spring, MD.

[20] S. Thomas and Y.T. Chan, "A Simple Approach for the Estimation of Circular Arc Center and Its Radius", Computer Vision, Graphics, and Image Processing 45, pp. 362-370, 1989.

[21] W.K. Pratt, "Digital Image Processing", Wiley, New York, 1977.

[22] Slama, "Manual of Photogrammetry", Fourth Edition, p. 1003, American Society of Photogrammetry, Virginia, 1980.

[23] R.Y. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses", IEEE Journal of Robotics and Automation, Vol. RA-3, No. 4, August 1987.

[24] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, "Numerical Recipes: The Art of Scientific Computing", Cambridge University Press, Cambridge, 1986.

[25] K. Preston Jr., "The Abingdon Cross Benchmark Survey", Computers, Volume 22, No.7, 1989.

[26] VDI (Verein Deutscher Ingenieure) - Nachrichten, April 1989.

APPENDIX A

THE PROGRAM SOURCE CODE OF

THE LOCAL STATISTIC METHOD

```
#include "c:\pcplus\itex\itexpfg.h"
#include "c:\pcplus\itex\stdtyp.h"
#include <math.h>

LOCALST(x,y,sigmal,bright)

int x,y,sigmal,*bright;

/* Refined Filtering of Image Noise Using Local Statistics       */
/* Paper Jong-Sen Lee/Computer Graphics and Image Processing 15  */
/* 380-389 (1981)                                                */
/* input:  x,y-coordinates of center of 7*7 matrix               */
/*         sigmal-noise variance ( proposed 300 by Lee )         */
/* output: bright-noise filtered brightness value for pixel x,y  */

{

   int matrix[4][4];
   int i,j,k,l,add,sub,or,submean;
   int ii=0,jj=0;
   float mean,var,kfactor;
   float sum_mean=0.,sum_var=0.;

   for ( i=x-2 ; i<=x+2 ; i=i+2 ) {
       ii++;
       jj=0;

       for ( j=y-2 ; j<=y+2 ; j=j+2 ) {
           jj++;

/* call submatr3 subroutine - which calculates the submean */
/*                            of the moving 3*3 window     */
           SUBMATR3(i,j,&submean);

/* generation of submatrix 3*3 - matrix[x=1...x=3][y=1...y=3] */
           matrix[ii][jj] = submean; } }

/* call edge_or subroutine - to find the orientation of the edge */
/*                           in matrix 3*3                       */
/* for orientation index see subroutine edge_or                  */
   EDGE_OR(matrix,&or);

/* pattern calculation of mean and variance                      */
/* ATTENTION index for orientation different as proposed by Lee  */
   switch (or) {
   case 0:   for ( k=x-3 ; k<=x+3 ; k++ ) {
             for ( l=y-3 ; l<=y   ; l++ )
                 sum_mean=sum_mean+rpixel(k,l); }
             mean = sum_mean/28.;
             for ( k=x-3 ; k<=x+3 ; k++ ) {
             for ( l=y-3 ; l<=y   ; l++ )
                 sum_var = sum_var + pow( (rpixel(k,l)-mean),2.0 );
```

```
            }
          break;

case 1:   add=-1;
          for ( k=x-3 ; k<=x+3     ; k++ ) {
              add++;
          for ( l=y-3 ; l<=y+3-add ; l++ )
              sum_mean=sum_mean+rpixel(k,l); }
          mean = sum_mean/28.;
          add=-1;
          for ( k=x-3 ; k<=x+3     ; k++ ) {
              add++;
          for ( l=y-3 ; l<=y+3-add ; l++ )
              sum_var = sum_var + pow((rpixel(k,l)-mean),2.0 );
          }
          break;

case 2:   for ( k=x-3 ; k<=x     ; k++ ) {
          for ( l=y-3 ; l<=y+3   ; l++ )
              sum_mean=sum_mean+rpixel(k,l); }
          mean = sum_mean/28.;
          for ( k=x-3 ; k<=x     ; k++ ) {
          for ( l=y-3 ; l<=y+3   ; l++ )
              sum_var = sum_var + pow((rpixel(k,l)-mean),2.0);
          }
          break;

case 3:   add=-1;
          for ( k=x-3     ; k<=x+3 ; k++ ) {
              add++;
          for ( l=y-3+add ; l<=y+3 ; l++ )
              sum_mean=sum_mean+rpixel(k,l); }
          mean = sum_mean/28.;
          add=-1;
          for ( k=x-3     ; k<=x+3 ; k++ ) {
              add++;
          for ( l=y-3+add ; l<=y+3 ; l++ )
              sum_var = sum_var + pow((rpixel(k,l)-mean),2.0);
          }
          break;

case 4:   for ( k=x-3 ; k<=x+3 ; k++ ) {
          for ( l=y   ; l<=y+3 ; l++ )
              sum_mean=sum_mean+rpixel(k,l); }
          mean = sum_mean/28.;
          for ( k=x-3 ; k<=x+3 ; k++ ) {
          for ( l=y   ; l<=y+3 ; l++ )
             sum_var = sum_var + pow((rpixel(k,l)-mean),2.0);
          }
          break;

case 5:   add=7;
```

```
                for ( k=x-3     ; k<=x+3 ; k++ ) {
                    add--;
                for ( l=y-3+add ; l<=y+3 ; l++ )
                    sum_mean=sum_mean+rpixel(k,l); }
                mean = sum_mean/28.;
                add=7;
                for ( k=x-3     ; k<=x+3 ; k++ ) {
                    add--;
                for ( l=y-3+add ; l<=y+3 ; l++ )
                    sum_var = sum_var + pow((rpixel(k,l)-mean),2.0);
                }
                break;

   case 6:      for ( k=x   ; k<=x+3 ; k++ ) {
                for ( l=y-3 ; l<=y+3 ; l++ )
                    sum_mean=sum_mean+rpixel(k,l); }
                mean = sum_mean/28.;
                for ( k=x   ; k<=x+3 ; k++ ) {
                for ( l=y-3 ; l<=y+3 ; l++ )
                   sum_var = sum_var + pow((rpixel(k,l)-mean),2.0);
                }
                break;

   case 7:      add=7;
                for ( k=x-3 ; k<=x+3     ; k++ ) {
                    add--;
                for ( l=y-3 ; l<=y+3-add ; l++ )
                    sum_mean=sum_mean+rpixel(k,l); }
                mean = sum_mean/28.;
                add=7;
                for ( k=x-3 ; k<=x+3     ; k++ ) {
                    add--;
                for ( l=y-3 ; l<=y+3-add ; l++ )
                   sum_var = sum_var + pow((rpixel(k,l)-mean),2.0);
                }
                break;
                } /* end of switch */

       var = sum_var/28.0;

/* calculation of k = kfactor and the filtered value *bright     */
      kfactor = (var - sigmal)/(var + sigmal);
      *bright = (mean + kfactor * ( rpixel(x,y) - mean ) +0.5);
      /* +0.5 to simulate ROUND(x)      */
}     /* end of subroutine LOCALST */
```

```
SUBMATR3(x,y,submean)

int x,y,*submean;

/* Subroutine SUBMATR3 - called by localst                          */
/*                       calculates the submean of a 3*3 window     */
/*                       with center pixel x,y                      */
/* input:  x,y - location of window                                 */
/* output: submean - mean of window                                 */
{

   int i,j;
   float sum;

   sum=0;

   for ( i=x-1 ; i<=x+1 ; i++ ) {
   for ( j=y-1 ; j<=y+1 ; j++ )
       sum = sum + rpixel(i,j); }

   *submean = ( sum/9. ) +0.5; /* +0.5 to simulate ROUND(x) */
}      /* end of subroutine SUBMATR3 */


EDGE_OR(matrix,or)

int matrix[][4];
int *or;

/* Subroutine EDGE_OR - called by localst                           */
/*                      calculates edge orientation within a        */
/*                      3*3 matrix                                  */
/* edge_or uses Three-level Simple Masks as proposed by Lee         */
/* Paper Edge Detection by Compass Gradient Masks                   */
/* Guener S. Robinson / Computer Graphics and Image Processing      */
/*                      6 , 492-501 (1977)                          */
/* input:  matrix 3*3                                               */
/* output: orientation - index as in paper of Robinson              */
{   /* start of subroutine EDGE_OR */

       int grad[3],stack,istack,i;

  /* Index 0 - Gradient NORTH                                       */
       grad[0] =   matrix[1][1] + matrix[2][1] + matrix[3][1]
                 - matrix[1][3] - matrix[2][3] - matrix[3][3];
       stack  = grad[0];
       istack = 0;

  /* Index 1 - Gradient NORTHWEST                                   */
       grad[1] =   matrix[1][1] + matrix[2][1] + matrix[1][2]
                 - matrix[3][2] - matrix[2][3] - matrix[3][3];
  /* Index 2 - Gradient WEST                                        */
```

```
        grad[2] =   matrix[1][1] + matrix[1][2] + matrix[1][3]
                  - matrix[3][1] - matrix[3][2] - matrix[3][3];

    /* Index 3 - Gradient SOUTHWEST                                    */
        grad[3] =   matrix[1][2] + matrix[1][3] + matrix[2][3]
                  - matrix[2][1] - matrix[3][1] - matrix[3][2];

     /* find maximum abs(gradient)                                     */
        for ( i=1 ; i<=3 ; i++ ) {
            if ( fabs(grad[i]) > fabs(stack) ) {
               stack  = grad[i];
               istack = i; } }

    switch (istack) {
    case 0:   if (  fabs(matrix[2][1]-matrix[2][2])
                  <fabs(matrix[2][2]-matrix[2][3])  ) *or = 0;
              else *or = 4;
              break;

    case 1:   if (  fabs(matrix[1][1]-matrix[2][2])
                  <fabs(matrix[2][2]-matrix[3][3])  ) *or = 1;
              else *or = 5;
              break;

    case 2:   if (  fabs(matrix[1][2]-matrix[2][2])
                  <fabs(matrix[2][2]-matrix[3][2])  ) *or = 2;
              else *or = 6;
              break;

    case 3:   if (  fabs(matrix[1][3]-matrix[2][2])
                  <fabs(matrix[2][2]-matrix[3][1])  ) *or = 3;
              else *or = 7;
              break;
              } /* end of switch */
}   /* end of subroutine EDGE_OR */
```

APPENDIX B

THE PROGRAM SOURCE CODE OF

THE CONTOUR FOLLOWING ALGORITHM

```
#include "c:\pcplus\itex\itexpfg.h"
#include "c:\pcplus\itex\stdtyp.h"
#include <stdio.h>

#define OBJECTIN 0
#define BACKINT 255

BTRACE(x,dx,y,dy,displayl,display2,
          xbuff,ybuff,buffmax,bufflgth,berr)

int *xbuff,*ybuff,*bufflgth,*berr;
int x,dx,y,dy,displayl,display2,buffmax;

/* Boundary Tracing Algorithm - Extract Boundary Pixels  */
/* BTRACE                       of Binary Images         */
/* INPUT  x,dx,y,dy: object window coordinates           */
/*        displayl : 1 - boundary pixels are displayed   */
/*        display2 : 1 - boundary pixels coordinates     */
/*                       are written to file WBUFF.DAT   */
/*        buffmax  : max. array length                   */
/* OUTPUT xbuff    : boundary coordinates array with     */
/*        ybuff    : length <buffmax>, to be defined     */
/*                   in calling program                  */
/*        bufflgth : length of array xbuff,ybuff         */
/*                   ATTENTION array [ 0..bufflgth ]     */
/*                   # of boundary pixels is bufflgth+1  */

{ /* start of BTRACE */

   int xn,yn,xin,yin,i,buffl,err;

   *berr = 0;
   xn = x;
   yn = y;

   start: SEARCHINIT(xn,x+dx,yn,y+dy,&xin,&yin);
   xbuff[0]=xin;
   ybuff[0]=yin;
   i = 0;

   NEXTBO(xbuff,ybuff,i,&err);

   if ( err == 1 ) {
      if (( xbuff[i] == x+dx ) &&
          ( ybuff[i] == y+dy )) {
          *berr = 1;
          return; }
      else {
          xn = xbuff[i]+10;
          yn = ybuff[i]+10;
          goto start; } }
   i++;
```

```
   while ( ( ! ((xbuff[i] == xbuff[0]) &&
                (ybuff[i] == ybuff[0])) ) && ( i<buffmax ) ) {

          NEXTBO( xbuff,ybuff,i,&err);

          if ( err == 1 ) {
             if (( xbuff[i] == x+dx ) &&
                ( ybuff[i] == y+dy )) {
                *berr = 1;
                goto end; }
             else {
                xn = xbuff[i]+10;
                yn = ybuff[i]+10;
                goto start; } }
        i++; }

   buff1 = i-1;
   *buff1gth = buff1;
   if ( i >= buffmax ) {
      printf("ERROR ARRAY OVERFLOW IN BTRACE\n\n");
      *berr = 1;
      goto end; }
   if ( display1 == 1 ) DRAWBUFF(xbuff,ybuff,buff1);
   if ( display2 == 1 )    WBUFF(xbuff,ybuff,buff1);
   end:;

} /* end of BTRACE */


SEARCHINIT(x1,x2,y1,y2,xin,yin)

int x1,x2,y1,y2,*xin,*yin;

/* Subroutine SEARCHINIT - finds initial pixel to start */
/*                         boundary tracing             */
/* INPUT     x1,x2: search area x direction             */
/*           y1,y2: search area y direction             */
/* OUTPUT  xin,yin: coordinates of initial pixel        */

{
   int countx,county;

   *xin = -1;
   *yin = -1;

   for ( countx=x1 ; countx<=x2 ; countx++ ) {
       for( county=y1 ; county<=y2 ; county++ ) {
       if (( rpixel(countx,county) == OBJECTIN ) &&
          ( doe(countx,county) < 4 )             &&
          ( doe(countx,county) > 0 )) {
          *xin = countx;
          *yin = county;
```

```
            countx = x2;
            county = y2; }
            } /* end of county - loop */
         } /* end of countx - loop */

} /* end of SEARCHINIT */


NEXTBO(xbuff,ybuff,i,err)

int *xbuff, *ybuff,i,*err;

/* Subroutine NEXTBO - searchs for next boundary pixel */
/*                     in subroutine BTRACE            */
/* INPUT  xbuff,ybuff: array of boundary pixels        */
/*                  i: pointer to last found           */
/*                     boundary pixel                  */
/* OUTPUT xbuff,ybuff: NEXT BOundary pixel             */
/*                i+1: pointer to new boundary pixel   */
/*                err: (1) if no next boundary pixel   */
/*                     was found - (0) okay            */

{
   int d,dstack,a,b,k,j,control,size,double1,double2;

   k = i;
   size = 30;      /* stack size to avoid double pixels */
   dstack = 0;
   control = 0;
   *err = 0;

   for ( a=-1 ; a<=1 ; a++  ) {
       for ( b=-1 ; b<=1 ; b++  ) {

           if ( rpixel(xbuff[i]+a,ybuff[i]+b) == OBJECTIN ) {
              d=DOE(xbuff[i]+a,ybuff[i]+b);

              if (( d > dstack ) && ( d < 4 )) {
                 double1=0;

                 for ( j=i ; j>=i-size ; j-- ) {
                     if (( xbuff[j] == xbuff[i]+a ) &&
                         ( ybuff[j] == ybuff[i]+b ) ) {
                         double1=1; }
                     } /* end of j - loop */

                     if ( ! ( double1 == 1 ) ) {
                        dstack = d;
                        xbuff[i+1] = xbuff[i]+a;
                        ybuff[i+1] = ybuff[i]+b;
                        control = 1; }
```

```
                } /* end of ( d>=dstack ) condition */
              } /* end of ( int(x,y) == objectin ) condition */
            } /* end of b - loop */
        } /* end of a - loop */

        while (( control == 0 ) && ( k > 0 )) {
              dstack=0;
              k--;

              for ( a=-1 ; a<=1 ; a++  ) {
                  for ( b=-1 ; b<=1 ; b++  ) {

                      if (rpixel(xbuff[k]+a,ybuff[k]+b) == OBJECTIN)
                         {
                         d=DOE(xbuff[k]+a,ybuff[k]+b);

                         if (( d > dstack ) && ( d<4 )) {
                             double2=0;

                             for ( j=i ; j>=i-size ; j-- ) {
                                 if (( xbuff[j] == xbuff[k]+a ) &&
                                    ( ybuff[j] == ybuff[k]+b ) ) {
                                    double2=1; }
                                 } /* end of j - loop */

                            if ( ! ( double2 == 1 ) ) {
                               dstack = d;
                               xbuff[i+1] = xbuff[k]+a;
                               ybuff[i+1] = ybuff[k]+b;
                               control = 1; }

                           } /* end of ( d>=dstack ) condition */
                      } /* end of (int(x,y)==objectin) condition */
                    } /* end of b - loop */
                } /* end of a - loop */
            } /* end of while ( control == 0 ) - loop */

   if ( control == 0 ) *err=1;
} /* end of NEXTBO */


DOE (x,y)

int x,y;

/* Subroutine DOE - calculates the degree of */
/*                  exposure of pixel x,y    */

{
   int d=0;

   if ( rpixel(x+1,y) == BACKINT ) d++;
```

```
   if ( rpixel(x-1,y) == BACKINT ) d++;
   if ( rpixel(x,y+1) == BACKINT ) d++;
   if ( rpixel(x,y-1) == BACKINT ) d++;
   return (d);

} /* end of DOE */


DRAWBUFF(xarr,yarr,arrayl)

int *xarr,*yarr;
int arrayl;

/* Subroutine DRAWBUFF - displays pixel in array */
/*                       white on black screen   */
/* INPUT xarr,yarr: array of pixel coordinates   */
/*       arrayl   : length of array              */

{
   int i;

   sclear(0);
   for ( i = 0 ; i <= arrayl ; i++ )
       wpixel(xarr[i],yarr[i],255);

} /* end of DRAWBUFF */


WBUFF(xarr,yarr,arrayl)

int *xarr,*yarr;
int arrayl;

/* Subroutine WBUFF - writes pixel in a file */
/*                    called wbuff.dat       */
/* INPUT xarr,yarr: array of pixel coordinates   */
/*       arrayl   : length of array              */

{
   int i;
   FILE *fp, *fopen();

   if ( (fp = fopen("wbuff.dat","w") ) != NULL) {
      for ( i = 0 ; i <= arrayl ; i++ )
          fprintf(fp,"%d %d\n",xarr[i],yarr[i]);
      fclose(fp); }
   else
      printf("ERROR OPEN FILE WBUFF.DAT\n");

} /* end of WBUFF */
```

APPENDIX C

THE PROGRAM SOURCE CODE OF

THE AUTOMATED THRESHOLD SELECTION ALGORITHM

```
#include "c:\pcplus\itex\itexpfg.h"
#include "c:\pcplus\itex\stdtyp.h"
#include <math.h>
#include <stdio.h>
#include <graph.h>

long HISTVALS[255];
float P[255],H[255],H_[255],S[255],S_[255],DELTA[255];

THSELECT(x,dx,y,dy,ints,inte,display,thval)

int x,dx,y,dy,ints,inte,display;
int *thval;

/* Subroutine THSELECT - Automatic Threshold Selection */
/*                       using the Entropy of the Gray */
/*                       Level Histogram               */
/* THSELECT is based on the algorithm proposed by      */
/* G.Johannsen, J.Bille IEEE CH1801 1982, A Threshold  */
/* Selection Method using Information Measures         */
/* INPUT  x,dx,y,dy: window-coordinates of operation   */
/*             ints: intensity to start evaluation     */
/*             inte: intensity to end evaluation       */
/*          display: 0 - evaluation is not displayed   */
/*                   1 - evaluation is displayed       */
/* OUTPUT     thval: selected threshold                */

{
   float LK,LK_,LKSUB1,LKSUB1_;
   float STACK;
   float NFLOAT,HISTFL;
   int STACKTH;
   int i,j,k,l1,l2;
   int ii=0,ksum=0;
   long N;
   char c1;

   /* Generation of Histogram */
   histogram(x,y,dx,dy,1,1,0,HISTVALS);
   N=0; STACK=1;

   for ( i=0 ; i<=255 ; i++ ) N=N+HISTVALS[i];
   for ( j=0 ; j<=255 ; j++ ) {
       NFLOAT=N;
       HISTFL=HISTVALS[j];
       P[j]=HISTFL/NFLOAT; }

   H[-1]=0 ; H_[-1]=0 ;
   LK=0;
   if ( display == 1 ) {
      _clearscreen(_GCLEARSCREEN);
      printf("INTENSITY K      DELTA[k]\n\n"); }
```

```
for ( k=0 ; k<=254 ; k++ ) {
    /* computation of S[k] */
    LKSUB1=LK;
    LK=LK+P[k];
    H[k]=0;

    for ( l1=0 ; l1<=k ; l1++ ) {
        if ( P[l1]>0 ) {
           if (( P[l1]/LK ) <= 0 ) printf("ERROR LN(0) - l1");
           H[k]=H[k]+(P[l1]/LK)*log(P[l1]/LK); }
        } /* end of l1 - loop */

    H[k]=-H[k];
    if ( LKSUB1>0 ) {
       S[k]=H[k]-(LKSUB1/LK)*H[k-1]; }
    else S[k]=H[k];

    /* computation of S_[k] */
    LK_=1-LK;
    LKSUB1_=1-LKSUB1;
    H_[k]=0;

    for ( l2=k+1 ; l2<=255 ; l2++ ) {
        if ( P[l2]>0 ) {
           if (( P[l2]/LK_ ) <= 0 ) printf("ERROR LN(0) - l2");
           H_[k]=H_[k]+(P[l2]/LK_)*log(P[l2]/LK_); }
        } /* end of l2 - loop */

    H_[k]=-H_[k];
    if ( LKSUB1_>0 ) {
       S_[k]=H_[k-1]-(LK_/LKSUB1_)*H_[k]; }
    else S_[k]=H_[k-1];

    /* computation of DELTA[k] */
    DELTA[k]=S[k]+S_[k];

    /* find minimum of DELTA[k] */

    if ( ( k>ints ) && ( k<inte ) ) {
       if ( display==1 ) printf("%3d        %f\n",k,DELTA[k]);
       if ( DELTA[k]==0 ) {
          ii++;
          ksum=ksum+k; }
       if ( DELTA[k]<STACK ) {
          STACK=DELTA[k];
          STACKTH=k; }

       } /* end of k - condition */
    }    /* end of k - loop      */
```

```
    if ( ii >= 3 ) STACKTH=ksum/ii;
    *thval=STACKTH;

    if ( display == 1 ) {
        _clearscreen(_GCLEARSCREEN);
        printf("\nTHRESHOLD: %d \n",STACKTH);
        printf("\nDELTA MIN: %f \n\n\n\n",STACK);
        printf("Press key to continue\n");
        cl=getchar();
        cl=getchar();
        _clearscreen(_GCLEARSCREEN); }

} /* end of THSELECT */
```

APPENDIX D

THE PROGRAM SOURCE CODE OF

THE RADIUS ESTIMATION ALGORITHM

```
#include <math.h>

ARCESTIM(xarr,yarr,N,sigma,centerx,centery,radius,biasR)

int xarr[],yarr[];
int N;
float sigma;
float *centerx,*centery,*radius,*biasR;

/* Subroutine ARCESTIM - simple exact solution for     */
/*                       estimating the location of    */
/*                       a circular arc and its radius */
/* ARCESTIM is based on an algorithm proposed by       */
/* S.M.Thomas /Y.T.Chan A Simple Approach for the      */
/* Estimation of Circular Arc Center and its Radius.   */
/* Computer Vision,  Graphics and Image Processing 45  */
/* pp.  362-370, 1989.                                 */
/* INPUT  x,y: coordinates of points ( xarr , yarr )   */
/*          N: number of points                        */
/*      sigma: variance of Noise                       */
/* OUTPUT centerx,centery: coordinates of center       */
/*                 radius: radius of circle            */
/*                  biasR: bias                        */

{
   int i;
   float sumx=0,sumy=0,sumxy=0,sumx2y=0;
   float sumx2=0,sumx3=0,sumxy2=0,sumy2=0,sumy3=0;
   float a1,a2,b1,b2,c1,c2;
   float xavg,yavg,R,sigma2,bxavg,byavg,br2;

   for ( i=0 ; i<=N-1 ; i++ ) {
       sumx=sumx+xarr[i];
       sumy=sumy+yarr[i];
       sumxy=sumxy+(float)(xarr[i])*(float)(yarr[i]);
       sumx2y=sumx2y+(float)(xarr[i])*(float)(xarr[i])
                                     *(float)(yarr[i]);
       sumx2=sumx2+(float)(xarr[i])*(float)(xarr[i]);
       sumx3=sumx3+(float)(xarr[i])*(float)(xarr[i])
                                    *(float)(xarr[i]);
       sumxy2=sumxy2+(float)(xarr[i])*(float)(yarr[i])
                                     *(float)(yarr[i]);
       sumy2=sumy2+(float)(yarr[i])*(float)(yarr[i]);
       sumy3=sumy3+(float)(yarr[i])*(float)(yarr[i])
                                    *(float)(yarr[i]); }

   a1=2*(sumx*sumx-N*sumx2);
   a2=2*(sumx*sumy-N*sumxy);
   b1=a2;
   b2=2*(sumy*sumy-N*sumy2);
   c1=sumx2*sumx-N*sumx3+sumx*sumy2-N*sumxy2;
   c2=sumx2*sumy-N*sumy3+sumy*sumy2-N*sumx2y;
```

```
    /* calculation of Center Coordinates */
    xavg=(c1*b2-c2*b1)/(a1*b2-a2*b1);
    yavg=(a1*c2-a2*c1)/(a1*b2-a2*b1);
    *centerx=xavg;
    *centery=yavg;

    /* calculation of Radius */
    R=sqrt( (sumx2-2*sumx*xavg+N*xavg*xavg+sumy2
                    -2*sumy*yavg+N*yavg*yavg)/N );
    *radius=R;
          sigma2=sigma*sigma;

    /* calculation of BIAS */
    bxavg=( 4*sigma2*(2-2*N)*N*(sumxy*sumy-sumy2*sumx) )/
          ( a1*b2-a2*b1);
    byavg=( 4*sigma2*(2-2*N)*N*(sumxy*sumx-sumx2*sumy) )/
          ( a1*b2-a2*b1);
    br2=( N*sigma2-2*sumx*bxavg+2*N*xavg*bxavg+N*bxavg*bxavg
         +N*sigma2-2*sumy*byavg+2*N*yavg*byavg+N*byavg*byavg)/N;
    *biasR=br2/(2*R);

} /* end of subroutine ARCESTIM */
```

APPENDIX E

THE PROGRAM SOURCE CODE OF

THE IMAGE GENERATOR

```
#include "c:\pcplus\itex\itexpfg.h"
#include "c:\pcplus\itex\stdtyp.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <graph.h>
#include <stdlib.h>
#include <sys/timeb.h>
#include <time.h>
#include <math.h>

main()

/* Program IMAG_GEN                                          */
/* Circular Image Generation with Smoothing,                 */
/* System Noise Simulation and Object Noise ( Texture )      */
/* Simulation                                                */

{ /* start of IMAG_GEN */

   int stop,radius,center_x,center_y,intensity_p,background;
   int xw1,xw2,yw1,yw2,rad,new_intensity,i,j;
   int rang_e1,rang_e2,pixel_num,diff,old_intensity;
   int kbkit(void),cnt,es;
   long xc1,yc1;
   unsigned rand_seed;
   double rad2,xc2,yc2,zc2,rnd2,rnd21;
   float noise,mean_n,std_n,range_n,rand4;
   char c1;
   FILE *fp, *fopen();

   _clearscreen(_GCLEARSCREEN);
   INITSYS();

   printf("Program - IMAG_GEN\n\n");
   printf("Image Simulation for Circular Features.\n\n");
   printf("Simulation Operations: - Smoothing\n");
   printf("                       - Additive System Noise\n");
   printf("                       - Texture Simulation\n\n\n");
   printf("Circle Generator\n\n");

   printf("Coordinates of the center of the circle\n");
   printf("X: ");
   scanf("%d",&center_x);
   printf("Y: ");
   scanf("%d",&center_y);
   printf("\n\nRadius of the circle R: ");
   scanf("%d",&radius);
   printf("\n\nIntensity of the pixels INSIDE the circle: ");
   scanf("%d",&intensity_p);
   printf("\nIntensity of BACKGROUND pixels: ");
   scanf("%d",&background);
```

```
/* Circle generation */

sclear(background);
circle(center_x,center_y,radius,1,1,intensity_p);
flood(center_x,center_y,intensity_p);

xw1=(center_x-radius-50);
yw1=(center_y-radius-50);
xw2=(center_x+radius+50);
yw2=(center_y+radius+50);

printf("\n\nPress 99 to exit / 0 to continue: ");
scanf("%d",&stop);
if ( stop == 99 ) goto end;
_clearscreen(_GCLEARSCREEN);

/* Ramp Edge Generator */

printf("Ramp Edge Generator\n\n");
printf("\nEdge width: ");
scanf("%d",&pixel_num);
diff=abs((intensity_p-background)/(pixel_num-1));

/* Calculation of distance to center for every pixel */
/* of the window                                     */
for (i=xw1 ; i<=xw2 ; i++) {
    for (j=yw1 ; j<=yw2 ; j++) {
        xc1=(i-center_x);
        yc1=(j-center_y);
        xc2=(xc1*xc1);
        yc2=(yc1*yc1);
        zc2=(xc2+yc2);
        rad2=sqrt(zc2);
        rad=rad2;

        /* For every pixel next to radius */
        /* perform smoothing operation    */
        if ( rad >= (radius-pixel_num/2) &&
             rad <= (radius+pixel_num/2) ) {
           rang_e2= rad-(radius-pixel_num/2);
           new_intensity=(intensity_p+(diff*rang_e2));
           wpixel(i,j,new_intensity); }
        } /* end of yw - loop */
    } /* end of xw - loop */

printf("\n\nPress 99 to exit / 0 to continue: ");
scanf("%d",&stop);
if ( stop == 99 ) goto end;
_clearscreen(_GCLEARSCREEN);
```

```
/* Noise generator */

printf("Additive System Noise Generator using
        Normal Distribution\n\n");
printf("Mean of the distribution: ");
scanf("%f",&mean_n);
printf("\nStandard deviation of the distribution: ");
scanf("%f",&std_n);

range_n=(2*std_n);
es=32767;

/* rand_seed is generated by system clock      */
/* Subroutine SECOND passes unsigned rand_seed */

rand_seed = SECOND();

_clearscreen(_GCLEARSCREEN);
srand(rand_seed);
for (i=xw1 ; i<=xw2 ; i++) {
    for (j=yw1 ; j<=yw2 ; j++) {
        rnd2=0.00000000;
        for (cnt=1 ; cnt<=12 ; cnt++) {
            rnd21=rand();
            rnd2=(rnd2+(rnd21/32767));
            } /* end of cnt - loop */
        rand4=(rnd2-6);

        noise=((rand4*range_n/2)+(mean_n));
        old_intensity=(rpixel(i,j));
        new_intensity=(old_intensity+noise);

        /* clip intensity if out of range */
        if ( new_intensity < 0 )
           new_intensity = -new_intensity;
        if ( new_intensity >255)
           new_intensity = 510-new_intensity;

        wpixel(i,j,new_intensity);
    } /* end of xw - loop */
} /* end of yw - loop */

printf("\n\nPress 99 to exit / 0 to continue: ");
scanf("%d",&stop);
if ( stop == 99 ) goto end;
_clearscreen(_GCLEARSCREEN);
```

```
   /* Texture Simulation */

   printf("Object Texture Simulation\n\n");
   printf("Using Additive Normal Distributed Noise
           in Object area\n\n");
   printf("Mean of the distribution: ");
   scanf("%f",&mean_n);
   printf("\nStandard deviation of the distribution: ");
   scanf("%f",&std_n);

   range_n=(2*std_n);
   es=32767;

   rand_seed = SECOND();

   _clearscreen(_GCLEARSCREEN);
   srand(rand_seed);
   for (i=xw1 ; i<=xw2 ; i++) {
       for (j=yw1 ; j<=yw2 ; j++) {
           xc1=(i-center_x);
           yc1=(j-center_y);
           xc2=(xc1*xc1);
           yc2=(yc1*yc1);
           zc2=(xc2+yc2);
           rad2=sqrt(zc2);
           rad=rad2;

           /* For every pixel within radius */
           /* perform texture simulation */
           if ( rad < radius ) {
              rnd2=0.00000000;
              for (cnt=1 ; cnt<=12 ; cnt++) {
                  rnd21=rand();
                  rnd2=(rnd2+(rnd21/32767));
                  } /* end of cnt - loop */
              rand4=(rnd2-6);

              noise=((rand4*range_n/2)+(mean_n));
              old_intensity=(rpixel(i,j));
              new_intensity=(old_intensity+noise);

              wpixel(i,j,new_intensity); }
           } /* end of yw - loop */
       } /* end of xw - loop */

   _clearscreen(_GCLEARSCREEN);
   end: saveim(0,0,640,480,0,"SIMULATE.IMG","");
   if ( (fp = fopen("prn","w") ) != NULL ) {
      fprintf(fp,"NOISE %f\n",std_n);
      fclose(fp);
      } else printf(" File couldn't be opened \n");
} /* end of IMAG_GEN */
```

```
SECOND()
/* Subroutine - SECOND
/* Generation of the random seed by reading the  */
/* seconds of the system clock                   */
{
   long s;
   unsigned seed;
   long two_byte;
   struct timeb time_now;

   ftime(&time_now);
   s = time_now.time;
   two_byte = pow(2,16);
   seed  = (s%two_byte);
   return seed;
}
```

APPENDIX F

THE PROGRAM SOURCE CODE OF

THE CALIBRATION ALGORITHM

```
/* CALIBRATION of Vision Systems                          */
/* Method of Roger Y. Tsai                                */
/* Paper - A Versatile Camera Calibration                 */
/* Technique for High-Accuracy 3D Machine Vision          */
/* Metrology Using Off-the-Shelf TV Cameras and           */
/* Lenses, IEEE Journal of Robotics and Automation        */
/* Vol.RA-3, No.4, August 1987                            */

#include <stdio.h>
#include <math.h>
#include "c:\nrcp\nr.h"
#include "c:\nrcp\nrutil.h"

/* global variables definition */
float d_x,Tx;
float r4,r5,r6,r7,r8,r9;
float *xf, *yf, *xw, *yw, *zw;
int   pnum;

/* constants of camera and vision system */
#define DX 0.669291e-3 /* inch */    /* 17 um 1*1e-6 m */
#define DY 0.511811e-3 /* inch */    /* 13 um 1*1e-6 m */
#define NCX 510
#define NCY 492
#define NFX 640

/* center pixel coordinates of frame memory */
#define CX 320
#define CY 240

/* uncertainty factor of sampling */
#define SX 1.042

/* define free vector allocation size */
#define MMP 50
#define NNP 20


SGN(arg)
/* Subroutine SGN determines the sign of a float variable */
/* and returns 1 if var >= 0 otherwise 0 is returned      */

float arg;
{
   if ( fabs(arg)*arg/(arg*arg) >= 0 ) return 1;
   else return 0;
}
```

```
void CALIB(xf,yf,xw,yw,zw,pnum)

float *xf, *yf, *xw, *yw, *zw;
int   pnum;

/* Subroutine CALIBRATION                                          */
/* INPUT xf,yf    : screen or frame buffer coordinates             */
/*       xw,yw,zw : world coordinates of calibration pattern       */
/*       pnum     : number of calibration points                   */
/* OUTPUT file CALIB.DAT with calibration parameters               */

{ /* start of CALIB */

   float *Xd, *Yd, **aa, *bb, *xx, *yy, *ww;
   float d_y;
   float Ty_1r1,Ty_1r2,Ty_1r4,Ty_1r5,Ty_1Tx;
   float r_1,r_2,r_4,r_5,ri,rj;
   float sr,T2y,Ty;
   float r1,r2,r3;
   float Tz,f,kl,Tzopt,fopt,klopt;
   float distx,disty;
   float Xddist,Yddist;
   float Xfdist,Yfdist;
   float Xwdist,Ywdist,Zwdist;
   int   i,n,err1,s;
   FILE *fp, *fopen();

   Xd = vector(1,MMP);  /* Numerical Recipes Subroutines */
   Yd = vector(1,MMP);  /* vector,  matrix               */
   aa = matrix(1,MMP,1,NNP);
   bb = vector(1,MMP);
   xx = vector(1,NNP);
   yy = vector(1,MMP);
   ww = vector(1,MMP);

   d_x = DX*NCX/NFX;
   d_y = DY;

   /* computation of distorted image coordinate */
   for ( i=1 ; i<=pnum ; i++ ) {
       Xd[i] = d_x*(xf[i]-CX)/SX;
       Yd[i] = DY*(yf[i]-CY);
       }

   /* solve for first set of variables */
   for ( i=1 ; i<=pnum ; i++ ) {
       aa[i][1] = Yd[i]*xw[i];
       aa[i][2] = Yd[i]*yw[i];
       aa[i][3] = Yd[i];
       aa[i][4] = -Xd[i]*xw[i];
       aa[i][5] = -Xd[i]*yw[i];
       bb[i]    = Xd[i];
```

```
    }
n = 5;

EQSOLVE(aa,pnum,n,bb,xx);

Ty_1r1 = xx[1];
Ty_1r2 = xx[2];
Ty_1Tx = xx[3];
Ty_1r4 = xx[4];
Ty_1r5 = xx[5];

/* compute ABS ( Ty ) */
r_1 = Ty_1r1;
r_2 = Ty_1r2;
r_4 = Ty_1r4;
r_5 = Ty_1r5;

/* introduce c - matrix [  r_1 r_2 ; r_4 r_5 ] */
if ( ( ( r_1 == 0 ) && ( r_2 == 0 ) ) or
     ( ( r_4 == 0 ) && ( r_5 == 0 ) ) or
     ( ( r_1 == 0 ) && ( r_4 == 0 ) ) or
     ( ( r_2 == 0 ) && ( r_5 == 0 ) ) ) {

     /* if a whole row or column of martix c vanishes */
     if ( ( r_1 == 0 ) && ( r_2 == 0 ) )
        { ri=r_4; rj=r_5; err1++; };
     if ( ( r_4 == 0 ) && ( r_5 == 0 ) )
        { ri=r_1; rj=r_2; err1++; };
     if ( ( r_1 == 0 ) && ( r_4 == 0 ) )
        { ri=r_2; rj=r_5; err1++; };
     if ( ( r_2 == 0 ) && ( r_5 == 0 ) )
        { ri=r_1; rj=r_4; err1++; };
        if (  err1 == 1 ) T2y=1/( ri*ri + rj*rj );
        else printf("c - matrix double zero error");
   }

/* if NOT a whole row or column of martix c vanishes */
else {
     sr = r_1*r_1 + r_2*r_2 + r_4*r_4 + r_5*r_5;
     T2y = ( sr-sqrt(sr*sr-4*pow((r_1*r_5-r_4*r_2),2.0 )) )/
           ( 2*pow((r_1*r_5-r_4*r_2),2.0) );
     }

Ty = sqrt(T2y);

/* determine the sign of Ty */
/* use a point whose computer image coordinates is away */
/* from the center */
Xddist = Xd[pnum];
Yddist = Yd[pnum];
Xwdist = xw[pnum];
Ywdist = yw[pnum];
```

```
Zwdist = zw[pnum];
r1 = Ty_1r1 * Ty;
r2 = Ty_1r2 * Ty;
r4 = Ty_1r4 * Ty;
r5 = Ty_1r5 * Ty;
Tx = Ty_1Tx * Ty;
distx  = r1*Xwdist + r2*Ywdist + Tx;
disty  = r4*Xwdist + r5*Ywdist + Ty;

/* check SGN - procedure in C */
if ( ! ( ( SGN(distx) == SGN(Xddist) ) &&
         ( SGN(disty) == SGN(Yddist) ) ) ) Ty=-Ty;

/* compute the 3D rotation matrix R */
r1 = Ty_1r1 * Ty;
r2 = Ty_1r2 * Ty;
r4 = Ty_1r4 * Ty;
r5 = Ty_1r5 * Ty;
Tx = Ty_1Tx * Ty;

/* check SGN - procedure in C */
if ( SGN(r1*r4 + r2*r5) ) s=-1;
else s=1;
r3 = sqrt(1-r1*r1-r2*r2);
r6 = s*sqrt(1-r4*r4-r5*r5);

/* r 7 - 9 determined by outer product */
r7 = r2*r6 - r3*r5;
r8 = r3*r4 - r1*r6;
r9 = r1*r5 - r2*r4;

/* compute a pre-value for focal length f */
/* solve for second set of variables */
/*   f Tz                                */
for ( i=1 ; i<=pnum ; i++ ) {
    yy[i] = r4*xw[i] + r5*yw[i] + Ty;
    ww[i] = r7*xw[i] + r8*yw[i];
    }
for ( i=1 ; i<=pnum ; i++ ) {
    aa[i][1] = yy[i];
    aa[i][2] = -Yd[i];
    bb[i] = ww[i]*Yd[i];
    }
n  = 2;

EQSOLVE(aa,pnum,n,bb,xx);

f = xx[1];
Tz= xx[2];
if ( f < 0 ) {
   r3 = -r3;
   r6 = -r6;
```

```
   r7 = -r7;
   r8 = -r8;
   }

/* solve for second set of variables again */
/*   f Tz                                   */
for ( i=1 ; i<=pnum ; i++ ) {
    yy[i] = r4*xw[i] + r5*yw[i] + Ty;
    ww[i] = r7*xw[i] + r8*yw[i];
    }
for ( i=1 ; i<=pnum ; i++ ) {
    aa[i][1] = yy[i];
    aa[i][2] = -Yd[i];
    bb[i]    = ww[i]*Yd[i];
    }
    n = 2;

EQSOLVE(aa,pnum,n,bb,xx);

f = xx[1];
Tz= xx[2];

/* compute the exact solution for f,Tz,k1  */
k1 = 0;
OPTIMIZE(f,Tz,k1,&fopt,&Tzopt,&klopt);
if ( (fp = fopen("calib.dat","w") ) != NULL )
   {
   fprintf(fp,"%f\n",r1);
   fprintf(fp,"%f\n",r2);
   fprintf(fp,"%f\n",r3);
   fprintf(fp,"%f\n",r4);
   fprintf(fp,"%f\n",r5);
   fprintf(fp,"%f\n",r6);
   fprintf(fp,"%f\n",r7);
   fprintf(fp,"%f\n",r8);
   fprintf(fp,"%f\n",r9);
   fprintf(fp,"%f\n",Tx);
   fprintf(fp,"%f\n",Ty);
   fprintf(fp,"%f\n",Tzopt);
   fprintf(fp,"%f\n",fopt);
   fprintf(fp,"%f\n",klopt);
   close(fp);
   }
else
   printf("Error open file < calib.dat >");

free_vector(Xd,1,MMP); /* Numerical Recipes Subroutines */
free_vector(Yd,1,MMP); /* free_vector,  free_matrix     */
free_matrix(aa,1,MMP,1,NNP);
free_vector(bb,1,MMP);
free_vector(xx,1,NNP);
free_vector(yy,1,MMP);
```

```
   free_vector(ww,1,MMP);

} /* end of CALIB */


float EQSOLVE(aa,m,n,bb,xx)

float **aa, *bb, *xx;
int m,n;

/* Subroutine - EQSOLVE                                  */
/* Solves overdetermined set of linear algebraic */
/* equations using singular value decomposition  */
/* Numerical Recipes, Press,Flannery,Teukolsky  */
/* Vetterling                                            */

{
   #define MP 50
   #define NP 20

   int i,j;
   float *w, *x, *b, **a, **u, **v;
   float wmin,wmax;

   w = vector(1,NP);
   x = vector(1,NP);
   b = vector(1,MP);
   a = matrix(1,MP,1,NP);
   u = matrix(1,MP,1,NP);
   v = matrix(1,NP,1,NP);

   for ( i=1 ; i<=m ; i++ ) {
   for ( j=1 ; j<=n ; j++ ) {
       a[i][j] = aa[i][j];
       }
       }

   for ( i=1 ; i<=m ; i++ ) b[i] = bb[i];
   for ( i=1 ; i<=m ; i++ ) {
   for ( j=1 ; j<=n ; j++ ) {
       u[i][j] = a[i][j];
       }
       }

   svdcmp(u,m,n,w,v);  /* Numerical Recipes Subroutine */

   /* find maximum singular value */
   wmax=0.0;
   for ( j=1 ; j<=n ; j++ )
       if ( w[j] > wmax ) wmax=w[j];
```

```
   /* define "small" */
   wmin=wmax*(1.0e-6);

   /* zero the "small" singular values */
   for ( j=1 ; j<=n ; j++ ) {
       if ( w[j] < wmin ) w[j]=0.0;
       }

   svbksb(u,w,v,m,n,b,x);  /* Numerical Recipes Subroutine */
   for ( j=1 ; j<=n ; j++ ) xx[j] = x[j];

   free_vector(w,1,NP);
   free_vector(x,1,NP);
   free_vector(b,1,MP);
   free_matrix(a,1,MP,1,NP);
   free_matrix(u,1,MP,1,NP);
   free_matrix(v,1,NP,1,NP);

} /* end of EQSOLVE */


#define NDIM 3
#define FTOL 1e-30
#define sqr(a) ((a)*(a))

float func(optvar)
float optvar[];

{
   int   i;
   float part1,rsqr,num,den,part2,sum;

                sum = 0;
                for ( i=1 ; i<=pnum ; i++ ) {
                rsqr  = sqr((1/SX)*d_x*(xf[i]-CX))+
                        sqr(DY*(yf[i]-CY));
                part1 = DY*(yf[i]-CY) + DY*(yf[i]-CY)
                                        *optvar[3]*rsqr;
                num   = r4*xw[i] + r5*yw[i] + r6*zw[i] + Tx;
                den   = r7*xw[i] + r8*yw[i] + r9*zw[i] + optvar[2];
                part2 = (optvar[1]*num)/den;
                sum   = sum + fabs(part1-part2);
                } /* end of i - loop */
                return sum;

}  /* end of func */
```

```
void dfunc(optvar,df)
float optvar[],df[];

{
   int   i;
   float part1,rsqr,num,den,part2,sum;

                sum = 0;
                for ( i=1 ; i<=pnum ; i++ ) {
                num   = r4*xw[i] + r5*yw[i] + r6*zw[i] + Tx;
                den   = r7*xw[i] + r8*yw[i] + r9*zw[i] + optvar[2];
                part2 = num/den;
                sum   = sum + fabs(part2);
                } /* end of i - loop */
                df[1] = sum;    /* Derivation of optvar[1] */

                sum = 0;
                for ( i=1 ; i<=pnum ; i++ ) {
                num   = r4*xw[i] + r5*yw[i] + r6*zw[i] + Tx;
                den   = r7*xw[i] + r8*yw[i] + r9*zw[i] + optvar[2];
                part2 = (-optvar[1]*num)/(den*den);
                sum   = sum + fabs(part2);
                } /* end of i - loop */
                df[2] = sum;     /* Derivation of optvar[2] */

                sum = 0;
                for ( i=1 ; i<=pnum ; i++ ) {
                rsqr  = sqr((1/SX)*d_x*(xf[i]-CX))
                        +sqr(DY*(yf[i]-CY));
                part1 = DY*(yf[i]-CY)*rsqr;
                sum   = sum + fabs(part1);
                } /* end of i - loop */
                df[3] = sum;     /* Derivation of optvar[3] */

}  /* end of dfunc */

OPTIMIZE(f,Tz,kl,fopt,Tzopt,klopt)

float f,Tz,kl,*fopt,*Tzopt,*klopt;

/* Subroutine - OPTIMIZE                              */
/* Optimization by Conjugate Gradient Methods         */
/* Polak-Ribiere variant                              */
/* Numerical Recipes, Press,Flannery,Teukolsky        */
/* Vetterling                                         */

{
   int iter;
   float fret,*p;

   p=vector(1,NDIM);
   p[1]= f;
```

```
    p[2]= Tz;
    p[3]= kl;

    /* Numerical Recipes Subroutine */
    frprmn(p,NDIM,FTOL,&iter,&fret,func,dfunc);

    printf("\nIterations: %3d\n",iter);
    printf("Func. value at solution %14f\n",fret);
    *fopt  = p[1];
    *Tzopt = p[2];
    *klopt = p[3];

    free_vector(p,1,NDIM);

} /* end of OPTIMIZE */
```

APPENDIX G

THE INPUT PARAMETERS FOR EACH METHOD AND SIMULATION CASE

		Low Contrast Simulation	High Contrast Simulation
Threshold(a) Method:	Threshold	90	140
Threshold(b) Method:	Lowcut	77	115
	Highcut	103	165
Roberts Edge Filter:	Gradient Threshold	24	49
Variance(3) Method: Option 1	Variance Threshold	380	1500
	Lowcut	52	65
	Highcut	128	215
Variance(5) Method: Option 1	Variance Threshold	841	3500
	Lowcut	52	65
	Highcut	128	215
Local Statistic: Method Option 1	Threshold	90	140
	Lowcut	52	65
	Highcut	128	215

Bandwidth for Local Methods with Option 2: 10 Pixels